CognitiveArm: Enabling Real-Time EEG-Controlled Prosthetic Arm Using Embodied Machine Learning

Abdul Basit, Maha Nawaz, Saim Rehman, Muhammad Shafique
eBRAIN Lab, Division of Engineering New York University (NYU) Abu Dhabi Abu Dhabi, UAE
{abdul.basit, mzn2386, sr7849, muhammad.shafique}@nyu.edu

Abstract*—Efficient control of prosthetic limbs via non-invasive brain-computer interfaces (BCIs) requires advanced EEG processing capabilities—including pre-filtering, feature extraction, and action prediction—all performed in real-time on edge AI hardware. Achieving this level of real-time processing on resource-constrained edge devices presents significant challenges in balancing model complexity, computational efficiency, and latency. We present CognitiveArm, an EEG-driven, brain-controlled prosthetic system implemented on edge AI hardware, achieving real-time performance without compromising accuracy. The system integrates BrainFlow—an open-source library for EEG data acquisition and streaming—and optimized deep learning (DL) models for precise brain signal classification. By leveraging evolutionary search, we identify Pareto-optimal DL model configurations through hyper-parameter tuning, optimizer analysis, and window selection, analyzed individually and in ensemble configurations. We further apply model compression techniques like pruning and quantization to optimize these models for embedded deployment, balancing computational efficiency and accuracy. We collected EEG dataset and designed an annotation pipeline, enabling precise labeling of brain signals corresponding to specific intended actions, which forms the foundation for training our optimized deep learning (DL) models. Our CognitiveArm system also supports voice commands for seamless mode switching, enabling control of the prosthetic arm's 3 degrees of freedom (DoF). Running independently on embedded hardware, CognitiveArm ensures low latency and facilitates real-time interaction. We developed a full-scale prototype of the CognitiveArm, interfaced with the OpenBCI UltraCortex Mark IV EEG headset. Our evaluations demonstrate a significant improvement in accuracy, reaching up to 90%, for classifying three core actions (left, right, and stay idle). The integration of voice commands allows for multiplexed, variable movement, enabling multi-action control for various everyday tasks (e.g., handshake, cup picking). This enhances CognitiveArm's real-world performance for prosthetic control, demonstrating its potential as a practical solution for individuals requiring advanced prosthetic limb control.

I. Introduction

The integration of non-invasive brain-computer interfaces (BCIs) with prosthetic limbs holds significant potential for enhancing assistive technologies, particularly for individuals with upper limb amputations or paralysis. Traditional control methods, such as surface electromyography (sEMG), rely on residual muscle activity, which can be unreliable or absent in conditions like amyotrophic lateral sclerosis (ALS), spinal cord injuries, and brainstem strokes [1]. In such cases, sEMG-based systems often fail to provide effective control due to muscle atrophy and ankylosis (see Table I). Electroencephalography (EEG)-based BCIs offer a promising alternative by directly interpreting brain signals, enabling prosthetic control even in the absence of functional muscles [2], [3].

The research problem targeted in this paper is to design an intelligent prosthetic arm with real-time embodied ML functions for multi-action EEG classification and voice command integration, which require developing efficient DL models and low-cost prototype, while ensuring low-latency and high accuracy under stringent design constraints.

Condition	Impact on EMG Use	EEG as a Solution
ALS	Muscle atrophy limits residual EMG signals	EEG-based BCI can interpret brain signals directly [4], [5]
Spinal Cord Injury	Loss of voluntary muscle control below the injury	EEG can bypass muscle control pathways [6], [7]
Brainstem Stroke	Severe loss of motor control, leading to locked-in syndrome	EEG can control assistive devices using brain signals [8]
Multiple Sclerosis	Muscle spasticity and weakness reduce EMG effectiveness	EEG can offer more reliable control options [5]
Muscular Dystrophies	Progressive muscle degeneration limits EMG utility	EEG allows control through brain signals [4], [8]

TABLE I: Comparison of EMG and EEG effectiveness in various conditions

State-of-the-Art Solutions and Their Limitation: Current state-of-the-art systems often rely on powerful personal computers or cloud-based servers for real-time EEG processing, which limits portability and introduces latency unsuitable for prosthetic control [9]. While DL models such as convolutional neural networks (CNNs), long short-term memory (LSTM) networks, and transformers have shown promise in extracting temporal and spatial features from EEG signals [10]–[13], these prior works often suffer from several limitations:

- *Lack of Full System Integration*: Many existing works focus on EEG signal classification in isolation and are not integrated into functional prosthetic prototypes with limited actions [10]–[13].
- *Computational Demands*: The complexity of DL models can hinder deployment on embedded devices due to limited computational resources and memory constraints [14].
- *Latency and Real-Time Performance*: Timing and latency, critical factors for real-time applications, are not thoroughly investigated. Solutions utilizing cloud-based processing introduce unacceptable latency and raise privacy concerns due to the transmission of sensitive neural data [15].
- *Model Compression Trade-offs*: Techniques like model quantization and pruning have been explored to reduce computational load [14], but they often come at the cost of accuracy, which is crucial for precise prosthetic control [16], [17].

Our research focuses on developing an intelligent prosthetic arm system that integrates high-performance EEG signal processing with embedded DL to ensure real-time functionality. The system's core is a low-latency DL-based action prediction framework that predicts *Action Labels*, which correspond to the user's intention among three core actions which lead to a variable amount of change in the position of the arm, enabling precise low-level control while adhering to resource constraints. These core actions are used to perform different tasks such as grabbing objects, handshake, etc. *Noteworthy, full system*

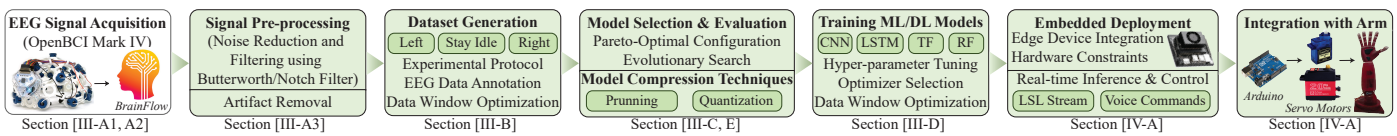


Fig. 1: Overview of the CognitiveArm system pipeline, including input acquisition, dataset generation, and Pareto-optimal model selection, and actuation.

integration is crucial to validate the efficiency of the DL-driven control loop, encompassing sensing, decision-making, motor control, and actuation in a synergistic way, enabling seamless real-world operation.

Research Challenges: These observations expose several research challenges and requirements (outlined below) in devising solutions for the targeted research problem. See overview in Figure 1.

- *Computational Complexity*: Developing DL models that are both accurate and efficient enough to run in real-time on resource-constrained edge devices.
- *Accuracy vs. Efficiency*: Investigating the impact of model compression techniques on accuracy and latency to ensure precise control without sacrificing significant performance.
- *EEG Dataset Generation and Annotation*: Creating a robust dataset generation and annotation pipeline to accurately capture and label EEG signals corresponding to specific intended actions.
- *Cost-effective Design*: Prototyping affordable prosthetic arms while ensuring functionality and robustness remains a key challenge, particularly for developing regions in the World.
- *Latency*: Ensuring low latency in the end-to-end system to facilitate real-time interaction and control.

Our Novel Contributions: To address these challenges, we introduce ***CognitiveArm*** that integrates non-invasive real-time EEG processing and classification using Embodied Learning with advanced DL architectures on edge AI processors for control of prosthetic limbs. To enable this, our CognitiveArm system employs the following:

- *An On-Device DL Engine* that leverages CNN, LSTM, Random Forest, and Transformer models individually and in an ensemble configuration for action prediction. This approach optimizes temporal and spatial feature extraction, ensuring high accuracy in EEG signal classification.
- *Efficient DL Model Design Space Exploration Method* employing an evolutionary search algorithm for identifying Pareto-optimal DL model configurations while balancing accuracy and efficiency trade-off. This process involves hyper-parameter tuning, optimizer selection, and window size selection to achieve the best trade-off between computational load and precision.
- *EEG Dataset Generation and Annotation Pipeline* to enable EEG data collection and a labeling protocol, ensuring precise and standardized annotation of brain signals corresponding to specific intended actions (left, right, stay idle) for each user.
- *Model Compression Techniques* like quantization and pruning to optimize the pareto-optimal DL model configurations for embedded deployment on a given edgeAI platform, while minimizing resource consumption and latency.
- *Full System Integration and Validation* using a full-functional in-house fabricated prosthetic arm prototype of *CognitiveArm*, while interfacing it with the OpenBCI UltraCortex Mark IV EEG headset. Our results demonstrate significant improvements in accuracy (up to 90%), response time, and system performance, underscoring the practicality of the system for real-world prosthetic control on NVIDIA Jetson Orin Nano.
- *Voice Command Integration* for seamless mode switching, enabling control of the prosthetic arm's 3 DoF and facilitating multi-action control for various everyday tasks (e.g., handshake, cup picking) along with model validation.

Paper Organization: Section II discusses related work. Section III presents the CognitiveArm system in further detail. Section IV details the Exploration Search, while Section V describes the experimental setup, followed by results and discussion in Section VI. We conclude in Section VII.

II. Related Work

Non-invasive brain-computer interfaces (BCIs) have become a focal point in prosthetic control due to their safety and accessibility compared to invasive methods. Invasive BCIs, like Neuralink's implanted electrodes, offer high-precision control but come with surgical risks and high costs [18]. Non-invasive systems, such as those developed by OpenBCI [19] and Emotiv [20], interpret EEG signals to control prosthetics and other devices [21]. However, these systems often face challenges with signal quality, limited channel counts, and the need for extensive user-specific calibration. Table II summarizes the performance of CognitiveArm against existing EEG-based prosthetic systems. CognitiveArm achieves higher accuracy (90%) and significantly lower latency due to model compression techniques (70% pruning, quantization). Unlike cloud-dependent systems, our on-device processing ensures real-time operation while maintaining computational efficiency.

TABLE II: Comparison of Brain-Controlled Prosthetic Arms

Solution	**Method**	**Acc.**[1]	**Cost**[2]	**Scope**
[22]	EEG-based	Mod.	Low	Limited real-time use
[23]	EEG-based	Mod.	High	Limited real-time use
[24]	EEG-based	Mod.	High	Power-intensive, limited use
[25]	EEG + sEMG	High	Mod.	High resource demand
[26]	EEG + EoG	80%	Mod.	Simple movements, user-dependent
[27]	EEG-based	High	High	Invasive Solution
[28]	EEG-based	87.5%	Low	Affordable, modular
[29]	EEG + sEMG	High	Low	Designed for developing regions
BeBionic [30]	sEMG-based	High	£30k	More Grips, fine motor control
LUKE Arm [31]	sEMG-based	High	$50k+	Powered joints, fine motor control
i-Limb [32]	sEMG-based	High	$40-50k	Multi-articulating, customizable
Michelangelo [33]	sEMG-based	High	$50k+	Advanced control, multiple grips
Shadow Hand [34]	sEMG-based	High	$65k+	High dexterity, advanced robotics
Congitive Arm	EEG-based	High	$500	3 DoF, efficient implementation

BCI technologies have shown potential in controlling prosthetic arms, providing mobility solutions for individuals with neurological or physical impairments [35]. Studies explore both invasive and non-invasive approaches to brain-controlled prosthetics [36], [37]. Several studies developed an EEG-based BCI system for prosthetic control aimed at ALS and spinal injury patients [22], [23]. However, they lack computationally efficient solution with simpler control methods described. Another research project on an EEG mind-controlled robotic arm [24] demonstrated hand movements, but suffered from high power consumption and inefficiencies. Further advancements have incorporated DL and 3D printing, such as in a cost-effective 3D-printed robotic arm controlled by sEMG sensors and DL [38]. *While promising, the model remains resource-hungry*. A separate work on brain-controlled prosthetic hands [26] demonstrated varied user-dependent accuracy (80%) for simple finger movements.

Machine learning techniques have also been explored in prosthetics. A comparison of KNN, SVM, and LDA classifiers for BCI control found LDA achieving up to 87.5% accuracy for movement classification [39]. A comprehensive review of BCIs in prosthetic control highlighted signal acquisition and real-time performance challenges. Innovative solutions like MindArm [28] and LIBRA NeuroLimb [29] emphasize affordability and accessibility in prosthetics. However, our work distinguishes itself by integrating a real-time, EEG-based control system optimized for low-latency and high-efficiency DNN. While these systems offer a cost-effective DNN-based solution, it lacks comprehensive integration of control

[1]**High Accuracy:** Classification accuracy exceeding 90% for all classes across test participants. **Moderate Accuracy:** Accuracy ranging between 75–90%, indicating improvable performance.

[2]**High Cost:** Products or services priced above $5000. **Moderate Cost:** Prices ranging between $1000–$5000, representing a mid-range category. **Low Cost:** Prices below $1000, signifying affordability.

efficiency and timing, which our system addresses to ensure precise and rapid response in real-world prosthetic applications.

III. CognitiveArm: Design and Methodology

The methodology for the *CognitiveArm* system encompasses the entire pipeline from EEG data acquisition to real-time prosthetic arm control. The key components include EEG signal acquisition and synchronization, data preprocessing and filtering, EEG dataset generation and annotation, model selection and optimization through evolutionary search, model training and evaluation, and integration with automatic speech recognition for enhanced control. The methodology of the CognitiveArm is illustrated in Figure 2.

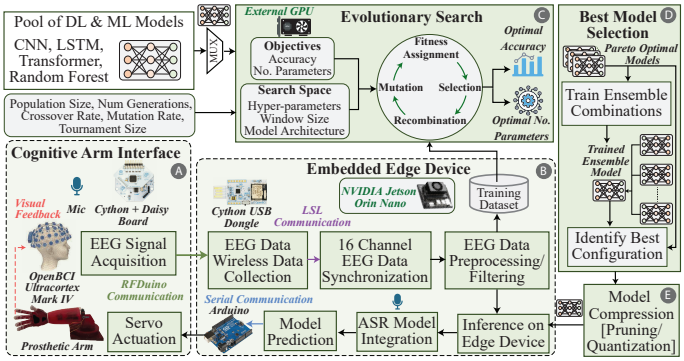


Fig. 2: *CognitiveArm* system methodology showcasing EEG signal acquisition, wireless data synchronization, preprocessing and filtering, evolutionary search for Pareto-optimal DL/ML models, ensemble training and best model selection, and real-time control of the prosthetic arm on an embedded edge device with integrated ASR-based voice commands.

A. EEG Data Acquisition and Preprocessing

1) EEG Data Acquisition: EEG data were acquired using the OpenBCI Ultracortex Mark IV EEG headset paired with 16-channel Cyton + Daisy Biosensing boards, which stream EEG data from 16 electrodes placed according to the 10-20 system (shown in Figure 3). This setup captures neural activity from key motor-related regions such as FP1, FP2, C3, and C4. The use of dry electrodes and multiple channels enables comprehensive, non-intrusive brain signal collection, ideal for DL feature extraction. Data acquisition was facilitated using BrainFlow [40], an open-source library optimized for high-fidelity EEG acquisition and streaming. BrainFlow's board-agnostic design and multi-threaded data streaming capabilities provide enhanced flexibility and efficiency, crucial for real-time control.

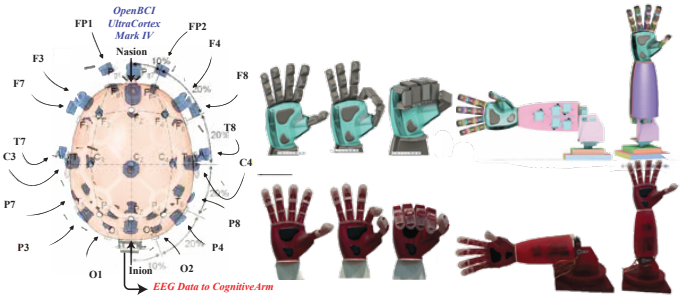


Fig. 3: EEG data is collected using the 10-20 electrode system, incorporating 16 electrodes for comprehensive brainwave acquisition. The recorded EEG signals are transmitted to the *CognitiveArm* for processing, where the system interprets the brain signals to control and demonstrate various hand poses.

2) Signal Acquisition and Synchronization: The EEG signals were collected and streamed using the Lab Streaming Layer (LSL) communication protocol [41], chosen for its low latency and high sample rate, which are critical for capturing rapid changes in brainwave patterns. LSL allows consistent data streaming at 125 Hz, ensuring precise synchronization and time-stamping, essential for accurate EEG signal acquisition in real-time setups. Figure 4 compares LSL with UDP, highlighting LSL's advantages in several key areas.

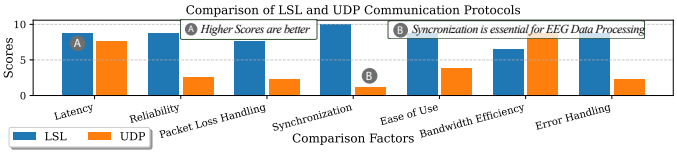


Fig. 4: Comparison of LSL and UDP, showing LSL's strengths in key factors, while UDP excels in bandwidth efficiency only.

3) Preprocessing and Filtering: To ensure high-quality EEG data for reliable prosthetic control, a series of preprocessing steps were implemented aimed at removing noise and artifacts:

- **Butterworth Bandpass Filter**: A 9th-order Butterworth bandpass filter was applied to retain the critical frequency range of 0.5–45 Hz, encompassing key brainwave activities (delta, theta, alpha, beta waves). This filtering process effectively removes low-frequency drift and high-frequency noise.
- **Notch Filter**: A 50 Hz notch filter with a quality factor of 30 was used to eliminate powerline interference, removing electrical noise from power sources that could contaminate the EEG signals.
- **Artifact Removal**: Standard signal cleaning techniques provided by BrainFlow were employed to address common EEG artifacts such as eye blinks and muscle movements, enhancing the signal-to-noise ratio (SNR) for accurate classification by the deep learning models. The result of filtering is illustrated in Figure 5.

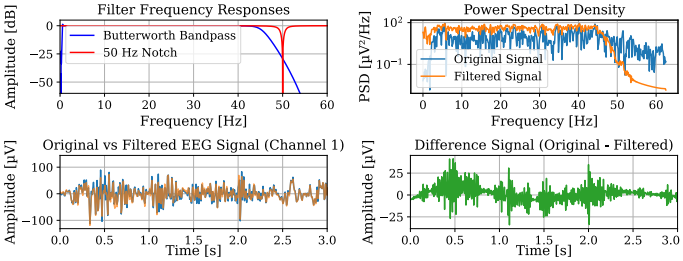


Fig. 5: Comparison of original and filtered EEG signals using a Butterworth bandpass filter and notch filter, demonstrating the removal of noise and retention of relevant brainwave frequencies for a single channel.

B. EEG Dataset Generation and Annotation

1) Experimental Protocol: Experiments involved five participants (four males and one female), aged between 18 and 31, all healthy individuals with no known neurological conditions. Participants were familiarized with the experimental setup and instructed to perform **mental tasks** corresponding to *three classes of actions* detailed below:

- *Move Right*: Participants imagined moving their right hand to the right without physical movement.
- *Move Left*: Participants imagined moving their left hand to the left without physical movement.
- *Stay Idle*: Participants maintained a calm state of mind without focusing on any particular movement.

Collection Structure: Each mental task was performed for 10 seconds, followed by a 10-second idle period, forming a structured sequence. This process was repeated to collect approximately 5 minutes of EEG data per participant across three separate sessions.

2) Data Labeling: EEG data were labeled based on auditory cues (beep sounds) indicating the start of each mental task. Labels were assigned for each mental task session as having one label, which was

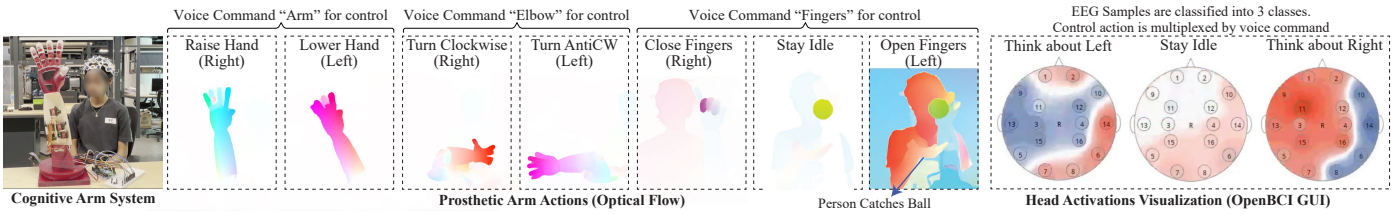


Fig. 6: Movement of the arm in correspondence the user's EEG signals is indicated. The optical flow images are generated using FlowFormer++ model [42].

later split into sliding windows each having the same label as the mental task. Potential delays between the auditory cue and participant response were accounted for by including transition periods in the labeled data. Participant feedback was obtained to confirm their mental focus during data collection, ensuring label accuracy.

3) Data Segmentation: The preprocessed EEG data were segmented into sliding windows to capture temporal dynamics:

- **Window Size**: Varying sizes ranging from 100 to 200 samples (0.8 s to 1.6 s at 125 Hz) were tested.
- **Sliding Step**: A step size of 25 samples (0.2 s) was used.

This resulted in a dataset suitable for input into the DL/ML models.

C. Model Selection and Optimization

1) Embodied DL/ML Model Selection: A set of DL/ML architectures was evaluated for EEG signal classification, including:

- Convolutional Neural Networks (CNNs) / Part of Ensemble
- Long Short-Term Memory (LSTM) networks / Part of Ensemble
- Transformers / Part of Ensemble
- Random Forest classifiers / Part of Ensemble

The goal was to balance classification accuracy with computational efficiency for real-time deployment on edge devices.

2) Evolutionary Search Algorithm: An evolutionary search algorithm was employed to identify Pareto-optimal model configurations, balancing performance and efficiency.

a) Problem Formulation: Let M represent the set of candidate models, each defined by architecture and hyperparameters. Models were optimized for two objectives: maximizing accuracy $A(m_i)$ and minimizing parameter count $P(m_i)$. The aim is to find models that offer the best trade-off between these objectives.

b) Algorithm Steps: We employ an evolutionary algorithm that operates on a population of models, evolving them over generations. The process is detailed below:

- **Initialization**: A population P_0 of N models with varied architectures and hyperparameters was generated randomly.
- **Fitness Evaluation**: Each model's fitness m_i was determined based on validation accuracy and parameter count, using a scoring function that balances these objectives with weights w_A, w_P:

$$S(m_i) = w_A \cdot \frac{A(m_i) - \min(A)}{\max(A) - \min(A)} - w_P \cdot \frac{P(m_i) - \min(P)}{\max(P) - \min(P)}$$

- **Selection**: Tournament selection was employed to choose parent models for reproduction based on fitness score $S(m_i)$.
- **Crossover and Mutation**: Evolutionary operations were applied to evolve the population over G generations, introducing diversity and preventing premature convergence.
- **Pareto Front Identification**: After G generations, models offering the best trade-off between accuracy and efficiency were identified. Pareto Front is evaluated using the following criteria:

$$F = \{m_i \in P_G \mid \nexists m_j \in P_G : A(m_j) > A(m_i) \ \& \ P(m_j) \leq P(m_i)\}$$

- **Best Model Selection**: Models meeting or exceeding an accuracy threshold α were selected based on minimal parameter count.

$$m_{\text{best}} = \begin{cases} \arg\min_{m_i \in F, A(m_i) \geq \alpha} P(m_i), & \text{if accuracy constraint met} \\ \arg\max_{m_i \in F} A(m_i), & \text{otherwise} \end{cases}$$

Algorithm 1 details the evolutionary search for model selection.

Algorithm 1 Evolutionary Search for Pareto-Optimal Model

Input: Population size N, generations G, accuracy threshold α, mutation rate p_m, crossover rate p_c
Output: Pareto-optimal model m_{best}
Initialize population P_0 of size N
for generation $g = 1$ to G **do**
  for each model $m_i \in P_g$ **do**
    Train and evaluate m_i to compute $A(m_i)$ and $P(m_i)$
    Compute fitness $S(m_i)$ based on accuracy and parameter count
  end for
  Select parents using tournament selection
  Apply crossover with probability p_c
  Apply mutation with probability p_m
  Update population P_{g+1} with offspring
end for
Compute Pareto front F from P_G
if $\exists m_i \in F$ such that $A(m_i) \geq \alpha$ **then**
  $m_{\text{best}} = \arg\min_{m_i \in F, A(m_i) \geq \alpha} P(m_i)$
else
  $m_{\text{best}} = \arg\max_{m_i \in F} A(m_i)$
end if
return m_{best}

c) Hyperparameter Tuning: Hyperparameters such as window size, learning rate, number of layers, hidden units, and dropout rates were optimized. Table III summarizes the hyperparameters and model architectures tested during the evolutionary search.

D. Model Training and Evaluation

1) Cross-Subject Validation: To evaluate model generalization, a leave-one-subject-out cross-validation strategy was employed. Data from four participants were used for training and validation (80:20 split), while the fifth participant's data was reserved for testing on unseen samples. Real-time validation was conducted with the prosthetic arm, where the DL models predicted control actions and translated them into corresponding motions. Ground truth labels for mental tasks were assigned based on vocal cues provided by participants during the session, ensuring accuracy in action validation. This process was repeated with each participant serving as the test subject, providing comprehensive validation across individuals.

2) Evaluation Metrics: The primary metric for model performance was classification accuracy, computed as the percentage of correctly classified windows over the total number of windows in the test set. To assess the statistical significance of the results, we calculated the mean accuracy and standard deviation across different test subjects.

3) Overfitting Analysis: To mitigate overfitting, we monitored training and validation losses to detect divergence and employed a leave-one-subject-out cross-validation strategy to ensure generalization to unseen participants. Regularization techniques, a balanced dataset, and a sliding window approach minimized bias across classes and participants. Additionally, the ensemble DL model enhanced

TABLE III: Hyperparameters and Model Architectures Tested in Evolutionary Search

Model	Architecture	Hyperparameters Tested	Optimizers	Other Details
LSTM	64, 128, 256 Units	Window Size (100–200), Dropout (0.1–0.5), Hidden Layers (1–3)	Adam, RMSProp	Learning Rate (1e-3–1e-5)
CNN	2–4 Conv Layers	Filter Sizes (3x3, 5x5), Pooling (Max/Avg), Stride (1–2)	Adam, SGD	Batch Size (32–128)
Random Forest	100–500 Trees	Features (Mean, Std, Min, Max, Var), Max Depth (10–None)	N/A (Non-Gradient)	Feature Selection
Transformer	2–6 Layers	Attention Heads (2–8), Hidden Units (64–256), Dropout (0.1–0.5)	AdamW	Weight Decay (1e-4–1e-6)

robustness and overall performance by aggregating predictions and reducing individual variances.

4) Ensuring Label Accuracy: To ensure label accuracy, several protocols were implemented. Transition periods were carefully incorporated to account for potential delays between auditory cues and participant responses, mitigating the effects of auditory lag. Before data collection, participants underwent a guided tutorial to familiarize themselves with the setup and the task requirements. The dataset was balanced across the three classes—*right*, *left*, and *idle*—to prevent model bias towards any particular action. Additionally, participants provided real-time feedback to confirm their mental focus during data collection, ensuring the reliability of the labels. These measures collectively enhanced the quality and consistency of the dataset.

E. Optimizing Neural Network Architectures

After identifying the best-performing models, model compression techniques were applied to optimize them for embedded deployment:

1) Pruning: Neural network connections were pruned at different levels (90%, 70%, 50%, 30%, and 0%), reducing the number of parameters without significant accuracy loss. Global pruning was applied to achieve a uniform reduction across the network.

2) Quantization: Post-training quantization converted model weights into lower-precision formats (e.g., 8-bit integers), decreasing memory usage and inference time. This step further optimized the models for deployment on resource-constrained edge devices.

F. Automatic Speech Recognition Integration

An Automatic Speech Recognition (ASR) system using the Whisper-small model [43] was integrated to enhance user interaction:

1) Voice Commands: The ASR system enabled voice commands for seamless mode switching, allowing users to control the prosthetic arm's degrees of freedom (e.g., "elbow," "arm," "fingers"). This facilitated multi-action control for various everyday tasks.

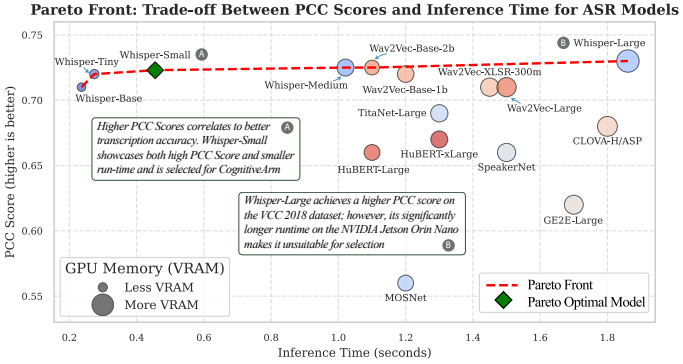


Fig. 7: Pareto front illustrating the trade-off between PCC scores and inference time for various ASR models. The size of each marker represents the VRAM usage (GPU memory) for each model. The Pareto front (red line) highlights the models offering the best balance between performance and resource efficiency, with the optimal model shown in green.

2) Voice Activity Detection: A Voice Activity Detection (VAD) algorithm was employed to trigger the ASR model only when speech was detected, minimizing resource consumption and latency.

3) Parallel Processing: The ASR system ran in a separate thread from the EEG classification, maintaining real-time responsiveness and ensuring that the EEG processing was not interrupted.

4) System Integration and Real-Time Control: The optimized models and ASR system were deployed on edge AI hardware, enabling the *CognitiveArm* system to operate independently with low latency, facilitating real-time interaction and control of the prosthetic arm. The end-to-end system integrates sensing, decision-making, motor control, and actuation, ensuring seamless operation in real-world scenarios.

IV. Experimental Setup

The experimental setup for cognitive Arm integrates several key components, including model training and deployment, signal processing, and hardware control mechanisms. The system is designed for real-time EEG-based control of prosthetic movements, enhanced by speech recognition capabilities. This section details the hardware, software, and models used in the experimental setup.

A. Prosthetic Arm Design and Prototyping

The prosthetic arm was designed using Autodesk Fusion 360, with three DoF: elbow flexion/extension and finger movements for gripping. The design was iteratively optimized for mechanical efficiency, durability, and functionality. Components were 3D printed using the Stratasys J750, ensuring precision and accurate material properties. The assembled arm features five embedded servo motors controlling finger movements for precise actions like gripping shown in Figure 3.

1) EEG Signal Acquisition and Processing: EEG signals are collected using the Ultracortex Mark IV headset with Cyton + Daisy boards, streaming data using BrainFlow for real-time inference. Preprocessing involves Butterworth bandpass filtering and a 50 Hz notch filter to remove noise and artifacts.

2) Model Training and Deployment: Neural network models (LSTM, CNN, Random Forest, and Transformer) are trained on the RTX A6000 Ada GPU to handle large training datasets efficiently. For edge deployment, the **NVIDIA Jetson Orin Nano** is used for real-time inference and prosthetic control, providing a balance of computational power and energy efficiency.

3) Generating Action Labels: Trained models classify EEG signals into action labels (e.g., move left, move right, stay idle) at 15 Hz. These labels are translated into motor control commands for the prosthetic arm.

4) Prosthetic Arm Control: Action labels are transmitted to an Arduino microcontroller, which sends precise motor control signals to servo motors, executing commands like gripping or elbow movement.

5) Real-World Validation: To validate the system in real-world scenarios, the EEG classification output was directly interfaced with the prosthetic arm. Participants independently controlled the arm's movements during test sessions, successfully translating their intended actions in 19 out of 20 sessions. For added reliability, participants verbally confirmed their intentions (e.g., saying *right* for a right-hand action), which was synchronized with EEG labels. This protocol minimized ambiguity and ensured consistent evaluation.

6) Servo Calibration: Servo motors are calibrated with a CCPM 3-channel tester to ensure alignment and consistent movement, optimizing the range of motion.

7) System Communication and Integration: The communication between the Jetson Orin Nano and the Arduino microcontroller is established via a serial communication protocol. The neural network models deployed on the Jetson Orin Nano process EEG signals and

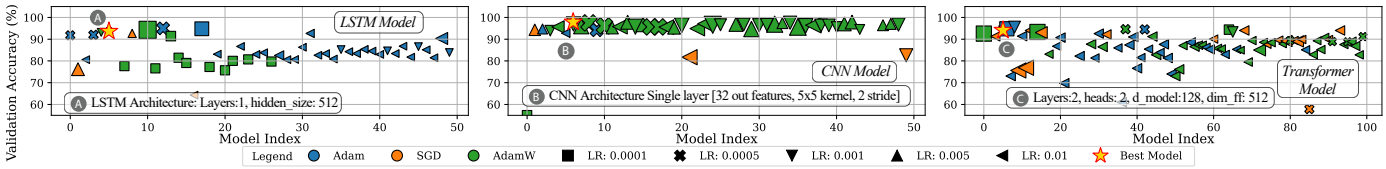


Fig. 8: Results highlight the selection of hyperparameters and model architecutes using evolutionary search, balancing accuracy and computational efficiency, and provides insights into the performance of CNN, LSTM, and Transformer models. The points highlighted are the pareto optimal models individually.

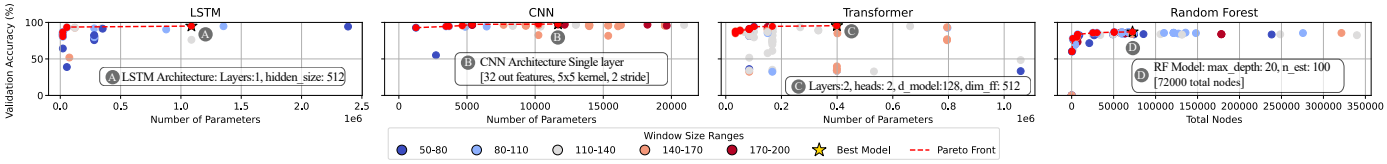


Fig. 9: Pareto front showcasing the trade-off between accuracy and parameter size for different model configurations. Models on the front achieve optimal balance, demonstrating the best accuracy given their parameter count. Particularly CNN models achieve high validation accuracy.

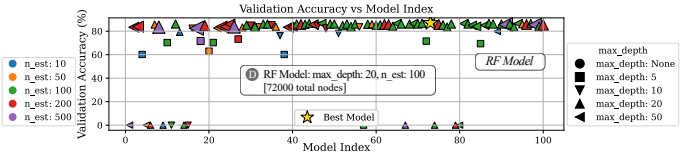


Fig. 10: Selection of hyperparameters using evolutionary search, balancing number of estimators and tree depth for Random Forest.

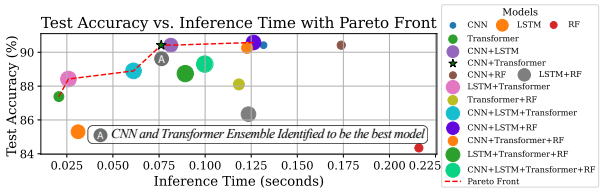


Fig. 11: comparing ensemble models based on their inference time and accuracy, showcasing the best-performing ensemble of CNN and Transformer models. The highlighted ensemble demonstrates the most optimal balance between quick response times and high accuracy, indicating its suitability for real-time applications.

generate action labels in real-time, which are then transmitted to the Arduino to control the prosthetic's motors, performing classification, and executing control commands.

8) Safety and Comfort Measures: Safety protocols were followed to ensure: Proper handling of the EEG headset and electrodes. Safe operation of the prosthetic arm, avoiding rapid or unexpected movements. Regular breaks between sessions to prevent fatigue and maintain concentration levels.

V. Results and Analysis

A. Statistical Analysis

To evaluate cross-subject variability, we normalized EEG data using the mean and standard deviation of each participant's readings. Statistical significance was assessed using paired t-tests to compare model performances across subjects. Additionally, confidence intervals (91%) were computed for the test accuracies of each model. For ensemble models, variance reduction was analyzed to confirm the robustness of predictions against user-specific noise.

We identified the best-performing models for EEG signal classification: a CNN with one convolutional layer (32 filters, kernel size 5, stride 2) using a window size of 190; an LSTM with a single layer of 512 hidden units and a window size of 130; a Transformer with 2 layers, 2 heads, a model dimension of 128, and a window size of 190; and a Random Forest with 200 estimators and a window size of 90. The results highlight the effectiveness of combining model

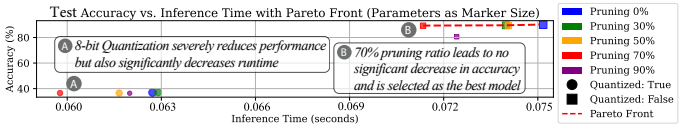


Fig. 12: Test Accuracy vs. Inference Time with Pareto Front. The 70% pruned model (B) maintains high accuracy (above 90%) with reduced runtime, while 8-bit quantization (A) significantly decreases runtime at the cost of accuracy. Marker size indicates model parameters.

architectures and compression techniques for EEG-driven prosthetic control. The CNN and Transformer ensemble achieved the best trade-off, with 91% accuracy and 0.075s inference time, ideal for real-time systems. Pruning further reduced inference time to 0.071 s while maintaining 90.1% accuracy, demonstrating its practicality for edge devices. Quantization improved latency to 0.036 s with significant accuracy reduction (38.5%), unsuitable for safety-critical applications. These findings validate the CognitiveArm system's efficiency and adaptability for resource-constrained environments.

VI. Conclusion

This paper presented an intelligent prosthetic arm system that integrates advanced EEG signal processing with embedded deep learning (DL) models, specifically optimized for real-time control and resource-constrained environments. Our approach leverages model compression techniques, including pruning and quantization, to significantly reduce latency and computational overhead while maintaining high accuracy. Through an evolutionary search method, we identified Pareto-optimal DL configurations that strike a balance between computational efficiency and predictive performance. The 70% pruned model demonstrated exceptional performance, achieving 90.1% accuracy while reducing inference time to enable real-time operation. The combination of architecture optimization, model compression, and robust evaluation across diverse participants ensures the system's adaptability to real-world applications. Furthermore, the proposed framework highlights the potential for scalable brain-computer interface (BCI) systems capable of delivering reliable, low-latency control for prosthetic devices. This work paves the way for future advancements in EEG-driven prosthetics, emphasizing efficiency, scalability, and practical deployment in everyday scenarios.

Acknowledgment

This work was partially supported by the NYUAD Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010.

REFERENCES

[1] P. Fitzgibbons and G. Medvedev, "Functional and clinical outcomes of upper extremity amputation," *JAAOS-Journal of the American Academy of Orthopaedic Surgeons*, vol. 23, no. 12, pp. 751–760, 2015.

[2] N. Padfield, K. Camilleri, T. Camilleri, S. Fabri, and M. Bugeja, "A comprehensive review of endogenous eeg-based bcis for dynamic device control," *Sensors 2022, Vol. 22, Page 5802*, vol. 22, p. 5802, 8 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/15/5802/htm https://www.mdpi.com/1424-8220/22/15/5802

[3] Y. Wang, M. Nakanishi, and D. Zhang, "Eeg-based brain-computer interfaces," *Advances in Experimental Medicine and Biology*, vol. 1101, pp. 41–65, 2019.

[4] Z. Z. Ma, J. J. Wu, Z. Cao, X. Y. Hua, M. X. Zheng, X. X. Xing, J. Ma, and J. G. Xu, "Motor imagery-based brain–computer interface rehabilitation programs enhance upper extremity performance and cortical activation in stroke patients," *Journal of NeuroEngineering and Rehabilitation*, vol. 21, pp. 1–14, 12 2024. [Online]. Available: https://jneuroengrehab.biomedcentral.com/articles/10.1186/s12984-024-01387-w

[5] "Trial: Non-invasive bci-controlled assistive devices — als tdi." [Online]. Available: https://www.als.net/als-trial-navigator/500010/

[6] J. Leng, X. Yu, C. Wang, J. Zhao, J. Zhu, X. Chen, Z. Zhu, X. Jiang, J. Zhao, C. Feng, Q. Yang, J. Li, L. Jiang, F. Xu, and Y. Zhang, "Functional connectivity of eeg motor rhythms after spinal cord injury," *Cognitive Neurodynamics*, pp. 1–15, 6 2024. [Online]. Available: https://link.springer.com/article/10.1007/s11571-024-10136-7

[7] H. Li, H. Ji, J. Yu, J. Li, L. Jin, L. Liu, Z. Bai, and C. Ye, "A sequential learning model with gnn for eeg-emg-based stroke rehabilitation bci," *Frontiers in Neuroscience*, vol. 17, p. 1125230, 4 2023.

[8] O. Maslova, Y. Komarova, N. Shusharina, A. Kolsanov, A. Zakharov, E. Garina, and V. Pyatin, "Non-invasive eeg-based bci spellers from the beginning to today: a mini-review," *Frontiers in Human Neuroscience*, vol. 17, p. 1216648, 8 2023.

[9] K. Belwafi, S. Gannouni, and H. Aboalsamh, "Embedded brain computer interface: State-of-the-art in research," *Sensors*, vol. 21, no. 13, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/13/4293

[10] W. H. Elashmawi, A. Ayman, M. Antoun, H. Mohamed, S. E. Mohamed, H. Amr, Y. Talaat, and A. Ali, "A comprehensive review on brain–computer interface (bci)-based machine and deep learning algorithms for stroke rehabilitation," *Applied Sciences*, vol. 14, no. 14, 2024.

[11] P. Samal and M. F. Hashmi, "Role of machine learning and deep learning techniques in eeg-based bci emotion recognition system: a review," *Artificial Intelligence Review*, 02 2024.

[12] A. Craik, Y. He, and J. L. Contreras-Vidal, "Deep learning for electroencephalogram (EEG) classification tasks: a review," *Journal of Neural Engineering*, vol. 16, no. 3, p. 031001, Feb. 2019.

[13] Y. Roy, H. Banville, I. Albuquerque, A. Gramfort, T. H. Falk, and J. Faubert, "Deep learning-based electroencephalography analysis: a systematic review," *Journal of Neural Engineering*, vol. 16, no. 5, p. 051001, aug 2019. [Online]. Available: https://dx.doi.org/10.1088/1741-2552/ab260c

[14] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017.

[15] A. Rodríguez, M. Z. Islam, A. S. M. S. Sagar, and H. S. Kim, "Enabling pandemic-resilient healthcare: Edge-computing-assisted real-time elderly caring monitoring system," *Applied Sciences 2024, Vol. 14, Page 8486*, vol. 14, p. 8486, 9 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/18/8486/htm https://www.mdpi.com/2076-3417/14/18/8486

[16] M. A. Hanif, G. M. Sarda, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Conlocnn: Exploiting correlation and non-uniform quantization for energy-efficient low-precision deep convolutional neural networks," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

[17] H. Ahmad, T. Arif, M. A. Hanif, R. Hafiz, and M. Shafique, "Superslash: A unified design space exploration and model compression methodology for design of deep learning accelerators with reduced off-chip memory access volume," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4191–4204, 2020.

[18] E. Musk and Neuralink, "An integrated Brain-Machine interface platform with thousands of channels," *J Med Internet Res*, vol. 21, no. 10, p. e16194, Oct. 2019.

[19] OpenBCI. (2024) The complete ultracortex. [Online]. Available: https://shop.openbci.com/products/the-complete-headset-eeg

[20] Emotiv. (2024) Epoc. [Online]. Available: www.emotiv.com/epoc

[21] N. Veena and N. Anitha, "A review of non-invasive bci devices," *Int. J. Biomed. Eng. Technol*, vol. 34, no. 3, pp. 205–233, 2020.

[22] H. A. Ali, N. Goga, C. V. Marian, and L. A. Ali, "An investigation of mind-controlled prosthetic arm intelligent system," *eLearning and Software for Education Conference*, pp. 17–26, 2020.

[23] O. Chinbat and J. S. Lin, "Prosthetic arm control by human brain," *Proceedings - 2018 International Symposium on Computer, Consumer and Control, IS3C 2018*, pp. 54–57, 7 2018.

[24] T. Beyrouthy, S. K. A. Kork, J. A. Korbane, and A. Abdulmonem, "Eeg mind controlled smart prosthetic arm," *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies, EmergiTech 2016*, pp. 404–409, 11 2016.

[25] D. Lonsdale, L. I. Zhang, and R. Jiang, "3d printed brain-controlled robot-arm prosthetic via embedded deep learning from semg sensors," *Proceedings - International Conference on Machine Learning and Cybernetics*, vol. 2020-December, pp. 247–253, 12 2020.

[26] X. Zhang, T. Zhang, Y. Jiang, W. Zhang, Z. Lu, Y. Wang, and Q. Tao, "A novel brain-controlled prosthetic hand method integrating ar-ssvep augmentation, asynchronous control, and machine vision assistance," *Heliyon*, vol. 10, p. e26521, 3 2024.

[27] M. Vilela and L. R. Hochberg, "Applications of brain-computer interfaces to the control of robotic and prosthetic arms," *Handbook of clinical neurology*, vol. 168, pp. 87–99, 1 2020. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/32164870/

[28] M. Nawaz, A. Basit, and M. Shafique, "Mindarm: Mechanized intelligent non-invasive neuro-driven prosthetic arm system," 2024. [Online]. Available: https://arxiv.org/abs/2403.19992

[29] A. A. Cifuentes-Cuadros *et al.*, "The libra neurolimb: Hybrid real-time control and mechatronic design for affordable prosthetics in developing regions," *Sensors*, vol. 24, no. 1, 2024.

[30] Ottobock, "Bebionic hand." [Online]. Available: https://www.ottobock.com/en-us/product/8E7*

[31] Mobiusbionics, "Luke arm." [Online]. Available: https://mobiusbionics.com/luke-arm/

[32] Ossur, "i-ilimb." [Online]. Available: https://www.ossur.com/en-us/prosthetics/arms/i-limb-quantum

[33] Ottobock, "Michelangelo hand." [Online]. Available: https://www.ottobock.com/en-ex/product/8E500

[34] S. RObot, "Shadow hand."

[35] J. Chen, Y. Xia, X. Zhou, E. V. Rosas, A. Thomas, R. Loureiro, R. J. Cooper, T. Carlson, and H. Zhao, "fnirs-eeg bcis for motor rehabilitation: A review," *Bioengineering 2023, Vol. 10, Page 1393*, vol. 10, p. 1393, 12 2023. [Online]. Available: https://www.mdpi.com/2306-5354/10/12/1393/htm https://www.mdpi.com/2306-5354/10/12/1393

[36] A. Kumari and D. R. Edla, "A study on brain–computer interface: Methods and applications," *SN Computer Science*, vol. 4, pp. 1–7, 3 2023. [Online]. Available: https://link.springer.com/article/10.1007/s42979-022-01515-0

[37] Q. He, Y. Yang, P. Ge, S. Li, X. Chai, Z. Luo, and J. Zhao, "The brain nebula: minimally invasive brain–computer interface by endovascular neural recording and stimulation," *Journal of NeuroInterventional Surgery*, vol. 0, pp. 1–7, 2 2024. [Online]. Available: https://jnis.bmj.com/content/early/2024/02/22/jnis-2023-021296 https://jnis.bmj.com/content/early/2024/02/22/jnis-2023-021296.abstract

[38] H. E. Abbady *et al.*, "3d-printed prostheses in developing countries: A systematic review," *Prosthetics and Orthotics International*, vol. 46, pp. 19–30, 2 2022.

[39] S. Rasheed, "A review of the role of machine learning techniques towards brain–computer interface applications," *Machine Learning and Knowledge Extraction*, vol. 3, no. 4, pp. 835–862, 2021. [Online]. Available: https://www.mdpi.com/2504-4990/3/4/42

[40] A. Parfenov *et al.*, "Brainflow." [Online]. Available: https://github.com/brainflow-dev/brainflow

[41] C. Kothe *et al.*, "The lab streaming layer for synchronized multimodal recording," *bioRxiv*, 2024.

[42] X. Shi, Z. Huang *et al.*, "Flowformer++: Masked cost volume autoencoding for pretraining optical flow estimation," 2023. [Online]. Available: https://arxiv.org/abs/2303.01237

[43] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," 2022. [Online]. Available: https://arxiv.org/abs/2212.04356

SQ-DM: Accelerating Diffusion Models with Aggressive Quantization and Temporal Sparsity

Zichen Fan[*‡], Steve Dai[†#], Rangharajan Venkatesan[†], Dennis Sylvester[*], Brucek Khailany[†]
[*]*University of Michigan, Ann Arbor, MI*, [†]*NVIDIA, Santa Clara, CA*
[‡]zcfan@umich.edu, [#]sdai@nvidia.com

***Abstract*—Diffusion models have gained significant popularity in image generation tasks. However, generating high-quality content remains notably slow because it requires running model inference over many time steps. To accelerate these models, we propose to aggressively quantize both weights and activations, while simultaneously promoting significant activation sparsity. We further observe that the stated sparsity pattern varies among different channels and evolves across time steps. To support this quantization and sparsity scheme, we present a novel diffusion model accelerator featuring a heterogeneous mixed-precision dense-sparse architecture, channel-last address mapping, and a time-step-aware sparsity detector for efficient handling of the sparsity pattern. Our 4-bit quantization technique demonstrates superior generation quality compared to existing 4-bit methods. Our custom accelerator achieves $6.91\times$ speed-up and 51.5% energy reduction compared to traditional dense accelerators.**

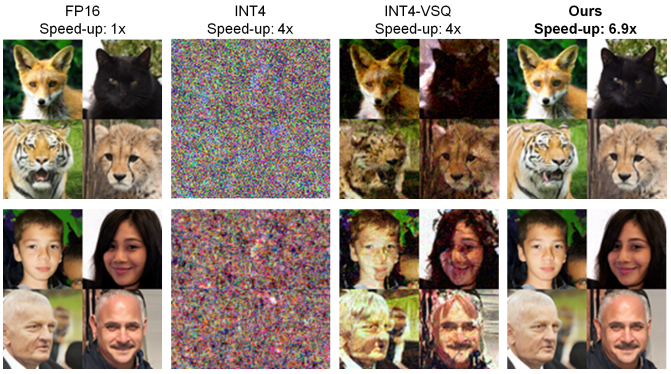


Fig. 1: Generated images and achieved speed-up for different data formats and quantization techniques.

I. Introduction

Diffusion models generate realistic contents by sequentially denoising data from random noise. They have emerged as the cornerstone of generative artificial intelligence (GenAI) for tasks such as image generation [1], video generation [2], and even weather prediction [3]. However, generating high-quality contents is notably slow because it requires repeatedly evaluating a large and complex neural network model over many time steps. Its complexity increases with higher resolution images and more advanced models, leading to large computation and memory overheads. Therefore, it is crucial to accelerate diffusion models to enable efficient deployment of real-world applications.

Quantization and sparsity serve as the two pivotal techniques driving dramatic improvements in deep neural network (DNN) performance over the last decade. Quantization enables reducing the precision of the weights and activations of a DNN, typically from 32-bit floating-point to low bitwidth formats like 4-bit integers. Sparsity refers to the practice of pruning less important weights and activations, such as zeroing out the two smallest values for every four adjacent values in a tensor. In isolation or combination, quantization and sparsity can lead to substantial improvements in compute and memory efficiency of DNN execution with smaller model sizes, faster computations, and lower power consumption.

While quantization and sparsity remain the first-order considerations when accelerating diffusion models, these techniques fail to work out-of-the-box due to the unique characteristics of these models. First, quantization error accumulates over time steps. Even small error in a single evaluation (time step) of the model becomes compounded over the many model evaluations required to generate a single sample (one image). Second, activation distributions in diffusion models vary across layers and time steps. This makes it difficult to apply a uniform quantization scheme over the entire generation process. Third, the data values within diffusion models tend to be extremely dense. The observed level of density precludes the effective use of sparse DNN accelerators for executing these models more efficiently.

In fact, we observe severe quality degradation when applying existing 4-bit data formats to the diffusion model. As shown in Figure 1, existing 4-bit formats such as `INT4` and `INT4-VSQ` result in obvious and unacceptable image quality degradation. In this paper, we propose novel software-hardware co-design techniques to accelerate diffusion models to achieve state-of-the-art generation quality at significantly reduced hardware costs. Our contributions are as follows:

- We are the first to aggressively quantize both weights and activations of diffusion models to 4-bit while promoting significant activation sparsity in these models.
- We propose a method to fully exploit the temporal nature of the sparsity during diffusion model sampling to maximize the efficiency of image generation.
- We present a new mixed-precision dense-sparse accelerator featuring channel-last addressing and temporal sparsity detection to effectively handle the sparsity pattern.

The rest of the paper is structured as follows: Section II provides preliminaries on diffusion models and discusses relevant work in quantizing and sparsifying these models. Section III presents optimizations we propose to improve diffusion model efficiency. Section IV details our accelerator implementation

to support those optimizations. Section V summarizes our findings and possible future directions.

II. Preliminaries

For the purpose of this work, we use Elucidated Diffusion Models, EDM [4] and EDM2 [5], as our baseline diffusion models. EDM is the strongest baseline available today in terms of training and sampling techniques because it is built by extensively exploring the design space for training and sampling for this type of model. More importantly, it is considered the state-of-the-art for its generation speed and quality, and thus very challenging to accelerate while maintaining good quality. EDM serves as the backbone model for applications such as text-to-image generation [1] and weather prediction [3], and generalizes to the family of convolution-based diffusion models.

Figure 2 illustrates the execution process and model architecture of EDM. Like most convolution-based diffusion models, EDM employs a U-Net based architecture with an encoder to capture features of the noisy input and a decoder to reconstruct the denoised output. As shown in Figure 2, EDM's architecture mainly consists of four types of layers: *Skip*, *Conv+SiLU*, *Embedding*, and *Attention*. The *Skip* block manages the skip connections from the encoder to the decoder, enabling information propagation from encoder to decoder and earlier layers to deeper ones. The *Embedding* linear layer transmits embedding information, including noise scheduling and labels (for conditional generation scenarios). The *Attention* block, present in specific layers (e.g., `enc.16x16_block_1` in the EDM1 model for CIFAR-10), implements an image self-attention mechanism to capture global context. Typically, dozens to hundreds of time steps (a.k.a. encoder-decoder model evaluations) must be executed to generate an image from random noise, as each time step returns a slightly denoised image from the previous time step.

A. Quantization

DNN quantization converts model weights and activations from high-precision floating-point down to low-precision data formats. Specifically, uniform symmetric quantization of a tensor $\mathbf{X}$ is expressed as $\hat{\mathbf{X}} = \text{round}(\mathbf{X}/s_x)$ where $s_x = \max(|\mathbf{X}|)/q_{\text{max}}$. Here $\mathbf{X}$ is the original tensor, $\hat{\mathbf{X}}$ is the quantized tensor, and s_x is the quantization scaling factor for $\mathbf{X}$. q_{max} represents the largest quantized value. The max operator can be taken at different granularity of $\mathbf{X}$, such as over the entire tensor, across each channel, or for each vector [6]. With scaled quantization, data will be stored and computed in the quantized format. Computation results need to be rescaled using the scale factors $\{s_x\}$ back to the model's original dynamic range.

Different techniques have been proposed to quantize diffusion model. PTQ4DM [7] and Q-diffusion [8] successfully quantize diffusion models to 8-bit. Several follow-up works [9]–[12] further improve their generation quality. Recently, SVDquant demonstrates 4-bit quantization on diffusion models using activation smoothing and low-rank decomposition [13]. However, it requires high-precision (FP16) low-rank branches and special kernel fusion tricks to alleviate the overhead of these branches. In contrast, we directly quantize both the original weights and activations to 4-bit with little quality degradation without smoothing or using any high-precision floating-point components. Our quantization technique also promotes sparsity simultaneously to enable further acceleration.

TABLE I: FID comparison of different models across datasets with various existing quantization formats.

Model, Dataset	EDM1, CIFAR-10	EDM1, AFHQv2	EDM1, FFHQ	EDM2, ImageNet
FP32	1.85	2.08	2.46	4.47
FP16	1.85	2.08	2.48	4.47
INT8	12.60	14.46	20.34	8.78
MXINT8 [22]	1.93	2.08	2.63	4.48
INT4	136.5	289.3	244.8	346.2
INT4-VSQ [6]	15.30	18.35	30.52	20.27

B. Sparsification

Sparsity in DNNs refers to the presence of zero-valued weights or activations, which can be exploited to reduce computational complexity and memory usage. In addition to quantization, we can prune less important weights and activations to create a sparse model that requires fewer resources. In fact, current-generation Tensor Cores provide support for structured weight sparsity that exploits a 2:4 (50%) sparsity pattern that enables twice the math throughput of dense matrix multiplications [14].

Previous works such as [15] and [16] have demonstrated that implementing structured weight sparsity in diffusion models can achieve the promised nearly 50% reduction in computation. These works leverage sparsity-aware finetuning and report modest degradation in model quality. Orthogonal to these efforts, we focus on activations instead of weights and promote sparsity that is not inherently present in the original model. We are able induce an average activation sparsity of 65% that leads to approximately 52% reduction in computational cost. Activation sparsity can be combined with weight sparsity to enable additional efficiency.

III. Diffusion Model Optimizations

To understand the landscape of quantizing diffusion models, we evaluate existing quantization techniques for EDM across various datasets (CIFAR-10 [17], AFHQv2 [18], FFHQ [19] and ImageNet [20]). Following EDM and previous work, we use Fréchet Inception Distance (FID) [21] to measure the quality of generated images: lower FID means better image quality. In our experiments, we generate 50,000 images in CIFAR-10, AFHQv2, FFHQ and 10,000 images in ImageNet to calculate the FID scores. Table I shows the results of the models in full-precision (FP32), half-precision (FP16), as well as variants of the most competitive and commonly accepted 8-bit and 4-bit formats. Here we use the same format for both weights and activations uniformly across the entire model.

From Table I, it is evident that `MXINT8` and `INT4-VSQ`, which employ fine-grained per-block scale factors, result in

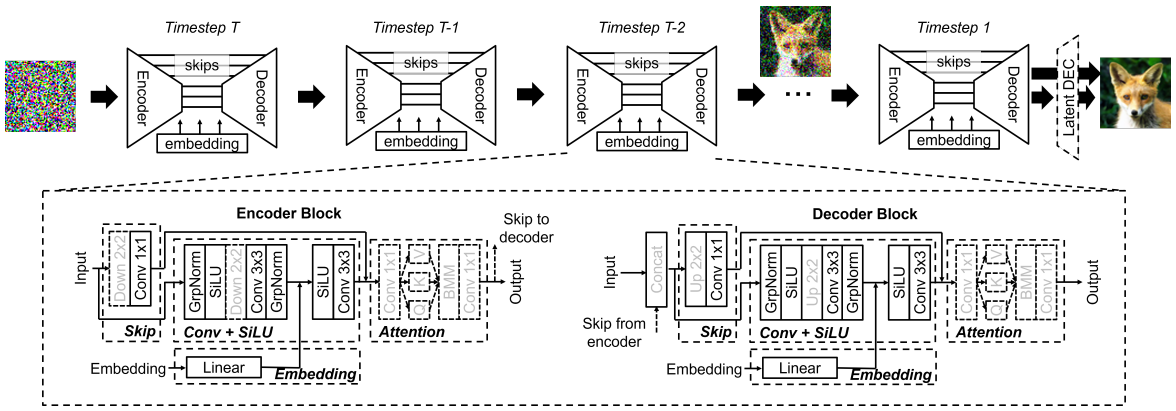


Fig. 2: Execution process and model architecture of EDM [4], [5].

significantly higher image quality compared to their INT8 and INT4 counterparts, which employ coarse-grained per-channel scale factors. Notably, the `MXINT8` configuration exhibits negligible degradation in image quality across all datasets. However, even with fine-grained scaling, the `INT4-VSQ` setting suffers from a substantial loss of image quality. While our exploration motivates the necessity of fine-grained scaling, it also suggests the need for new optimizations of the model and the data format to achieve baseline-level quality in low-precision. In the following sections, we assume fine-grained scaling exclusively for data formats.

A. *Mixed-precision Quantization*

The sensitivity to quantization across different layers or computation types is especially notable in these diffusion models, based on our block-wise quantization sensitivity experiment in Figure 3. In this experiment, we keep one block in 4-bit precision while all other blocks are set to 8-bit. While we simply use `MXINT8` for the 8-bit blocks, we specifically propose our own `INT4` format with `FP8` scale factors for the 4-bit blocks to improve dynamic range of the representation. The results in Figure 3 indicate that only the first and last few blocks are generally more sensitive to quantization. Consequently, it suffices to maintain these quantization-sensitive blocks at higher precision (`MXINT8`) while using 4-bit for the remaining blocks. The computation and memory cost of the high-precision blocks account for only about 5% of the total cost, allowing the system to retain significant benefits from the 4-bit precision.

Additionally, Figure 4 presents a breakdown of the computation and memory costs for the four types of blocks in the model. Notably, more than 90% of the total computation cost and 85% of the total memory cost are attributed to the *Conv+SiLU* block, highlighting its significance in the model's overall performance. Therefore, we emphasize quantizing the *Conv+SiLU* computation blocks to 4-bit, with other less important blocks in 8-bit. The result of these optimizations are presented in the `Ours(MP-only)` row of Table II, demonstrating appreciable improvement from those of the baseline data format while attaining 73% and 72% average reduction in both computation and memory cost, respectively. Here we

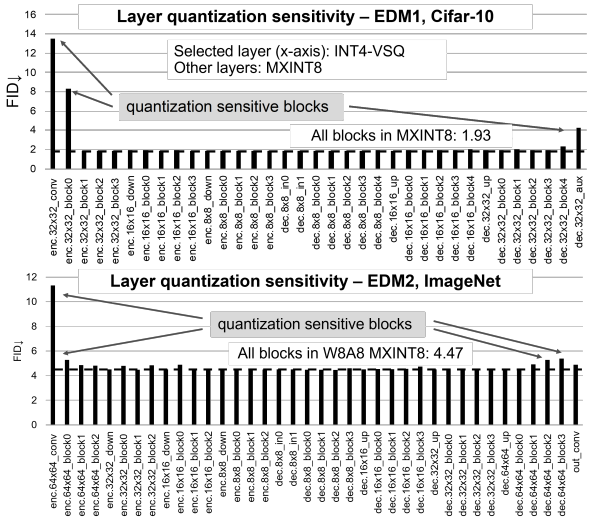


Fig. 3: Block-wise quantization sensitivity for EDM model.

assume a computational equivalence of 1 `FP16` multiplication to 2 `INT8` multiplications to 4 `INT4` multiplications based on computation resources and memory bandwidth [23].

B. *Hardware-efficient Activation Function*

Although our mixed-precision quantization technique significantly improves the FID score, we identify additional optimization opportunities with the non-linear activation functions (SiLU) prevalent in these models. Figure 5 (left) shows the

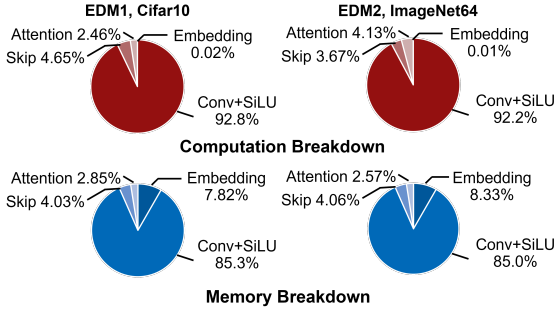


Fig. 4: EDM model computation and memory breakdown.

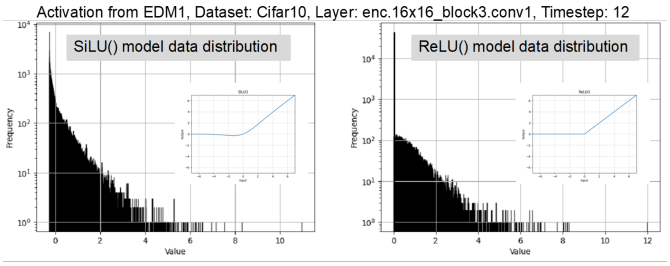


Fig. 5: Comparison of activation data distributions at the output of Conv+SiLU versus Conv+ReLU.

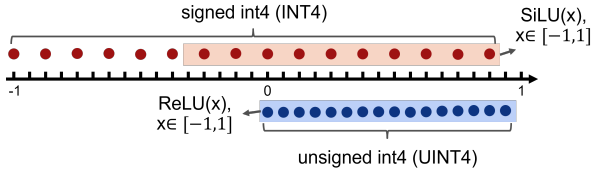


Fig. 6: SiLU(x) after `INT4` quantization versus ReLU(x) after `UINT4` quantization.

activation data distribution of one *Conv + SiLU* layer where $\text{SiLU}(x) = x/(1+e^{-x})$ [24]. At the output of the SiLU non-linear function, the data distribution spans from $[-0.278, \infty)$. The existence of a small negative range necessitates the use of signed data formats (e.g., signed `INT4`) for activations. Figure 6 (top) illustrates the signed `INT4` quantization levels needed when SiLU is used. When the input x is in the range $[-1, 1]$, the output of $\text{SiLU}(x)$ lies within $[-0.269, 0.731]$. Based on the quantization formula in Section II-A, only 10 of the 16 levels in signed `INT4` can be used, resulting in severe under-utilization of the available bitwidth.

In contrast, ReLU is a widely adopted and hardware-efficient non-linear function where the output data distribution ranges from $[0, \infty)$ [25]. With ReLU in place of SiLU, we can focus our representation on only the positive range and quantize the activations to unsigned `INT4` (`UINT4`) data format to achieve more precise representations. Figure 6 (bottom) depicts the quantization of ReLU activations when x is within $[-1, 1]$. All the available quantization levels of the `UINT4` format can be used. As a result, leveraging *Conv+ReLU* makes the model more quantization-friendly compared to using *Conv+SiLU*.

To adapt the SiLU-based model to use ReLU, we replace the non-linear activation function and then finetune the pre-trained SiLU-based model. The finetuning process takes less than 10% of the total pre-training time [4]. The resulting ReLU-based model achieves similar image quality to the original SiLU-based model. Figure 1 presents example images (targeting AFHQv2 and FFHQ datasets) generated by the SiLU-based models with existing quantization techniques (left three images) and ReLU-based models (the fourth image) with our techniques, as well as their respective performance. We can see that the 4-bit quantized ReLU-based model is clearly superior in quality to other 4-bit quantized models.

To derive the ReLU-based model, we finetune the full-precision version of the model followed by post-training quantization (PTQ) rather than directly performing quantization-aware training (QAT). This methodology avoids the overhead of QAT and produces a single ReLU-trained model that can be adapted to different quantization settings and hardware targets. However, we do expect QAT on the ReLU-based model to attain incremental quality gain. The FID score for the ReLU-based model with our proposed 4-bit quantization scheme is reported in the `Ours (MP+ReLU)` row in Table II. Indeed, by replacing the *Conv+SiLU* block with the *Conv+ReLU* block, our 4-bit diffusion model achieves the best FID scores.

TABLE II: FID comparison of different quantized models.

Quant Method	Avg. Comp. Saving	Avg. Mem. Saving	EDM1, CIFAR-10	EDM1, AFHQ v2	EDM1, FFHQ	EDM2, ImageNet
INT4-VSQ	75%	75%	15.60	18.35	30.52	20.27
Ours (MP-only)	73%	72%	2.87	2.39	5.41	8.75
Ours (MP+ReLU)	**73%**	**72%**	**2.12**	**2.35**	**3.10**	**6.93**

C. Temporal Per-channel Sparsity

In addition to enabling aggressive 4-bit quantization, the use of ReLU simultaneously promotes significant activation sparsity in the model, as ReLU clamps all negative values strictly to zero. The average sparsity of the SiLU-based model is around 10%, whereas the ReLU-based model achieves a significantly higher average sparsity of 65% and up to 85% sparse for some layers. However, mid-level random sparsity does not translate well to hardware acceleration; as shown in [26], unstructured sparsity requires at least 87.5% sparsity to yield notable speed-ups on GPUs.

Interestingly, instead of random sparsity, we observe a temporal per-channel sparsity pattern in ReLU-based diffusion models that can be effectively exploited for further acceleration. Figure 7 depicts the sparsity pattern of the activations of a single layer in a ReLU-based EDM (for CIFAR-10 dataset). The values are binarized: zero values are represented in black, and non-zero values in white. Each row corresponds to a channel (32x32 block) of the activation tensor, while each column represents a time step in the diffusion process.

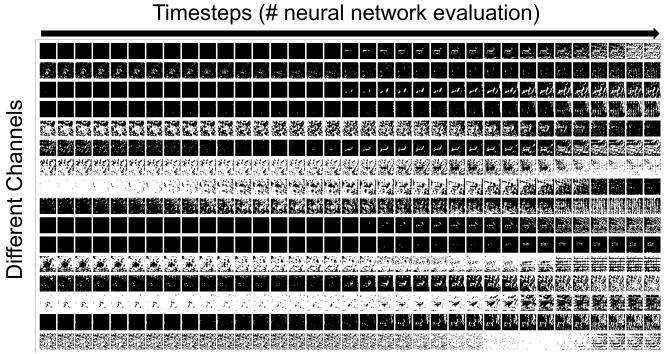


Fig. 7: Temporal per-channel sparsity pattern.

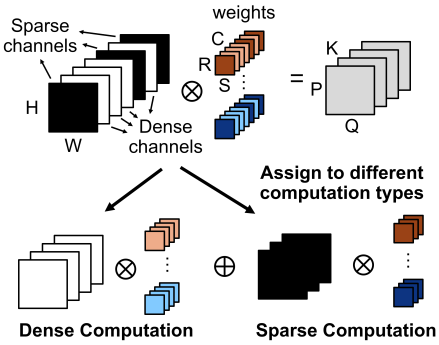


Fig. 8: New computation scheme for temporal per-channel sparsity pattern.

From Figure 7, it is evident that different channels exhibit distinct sparsity characteristics; some channels are highly sparse (with predominantly black pixels) while others are dense (with mostly white pixels). Furthermore, the sparsity of each individual channel varies across time steps; sparse channels can become dense and vice versa. This temporal per-channel sparsity presents a perfect opportunity for acceleration: sparse channels can be grouped and processed using a sparse engine, while dense channels can be handled by a dense engine. In the following section, we introduce a new hardware architecture to support our aggressively quantized and temporal sparse diffusion models.

IV. Diffusion Model Accelerator

In this section, we propose a novel computation scheme and hardware architecture to accelerate the temporal per-channel sparsity pattern observed in ReLU-based model activations. Figure 8 illustrates the core concept: the activation tensor is categorized into sparse and dense channels. By grouping channels (including both weights and activations) based on the activation sparsity type, the computation can be optimized. Please note that the weights are always dense in this work. Under the proposed scheme, dense channel groups are processed using the dense processing unit, while sparse channel groups are computed using the sparse processing unit. After that, the partial sums are added together to get the final result. In following sections, we detail the architecture that leverages temporal sparsity to accelerate diffusion model while significantly reducing total system energy consumption.

A. Overall Architecture

Figure 9 illustrates the overall architecture of our diffusion model accelerator design, adapted from the MAGNet modular accelerator generator [27]. The accelerator comprises of three main components: a controller, a dense/sparse processing element (D/S PE) array, and an interconnection network between the global buffer and the PEs. The controller manages time step information and orchestrates various PEs through control logic. The architecture features two types of PEs: Dense Processing Elements (DPEs) for dense channel computation and Sparse Processing Elements (SPEs) for sparse channel computation. These PEs are interconnected via configurable routers (R), enabling efficient processing of both dense and sparse data.

The D/S PE consists of several key components: a sparsity-aware address generator, weight/input/accumulation buffers, dense/sparse datapaths, and a post-processing unit (PPU) with a sparsity detector. Each PE can be configured to either the dense or sparse datapath, depending on the computation type. Based on the sparsity pattern, either the dense or sparse vector MAC datapath is employed to compute partial sums. Recent works on dense accelerators [28], [29] and sparse accelerators [30]–[37] have shown the effectiveness of tailored architectures for specific data types. In our design, we leverage a MAERI-like architecture [29] for the dense MAC datapath and a SIGMA-like architecture [34] for the sparse MAC datapath, as both architectures are well-suited for handling irregular matrix sizes. SIGMA's flexible distribution and reduction networks also enable efficient processing of irregular matrix sparsity.

After processing all input channels, the data from the accumulation buffer is passed through the PPU. The sparsity-aware address generator maintains channel information, including sparsity type (dense, sparse, etc.) and the corresponding channel index. Utilizing this information, the address generator produces the necessary weight and activation addresses to fetch data from the global buffer. A temporal sparsity detector in the PPU identifies per-channel sparsity in the output. The number of zeros is compared against a predefined threshold to update the sparse channel index information for the next layer in the sparsity-aware address generator.

B. Channel-last Memory Mapping

To accommodate the non-continuous input channel order required by the sparsity-aware address generator when fetching data from the global buffer, we design a channel-last data mapping strategy, illustrated in Figure 10. For activations, the address mapping follows the sequence of width (W), height (H), and input/output channel (C) being the last, enabling the PE to fetch the entire dense/sparse input/output channel. For sparse channels, only nonzero values and its binary indicator (0 for zero and 1 for non-zero) are stored in the memory, which aligns with the SIGMA sparse accelerator architecture. For weights, the mapping sequence is kernel width (S), kernel height (R), output channel (K), and then input channel (C). This channel-last ordering ensures that the corresponding weights are fetched to align with the activations, optimizing the computation process.

C. Temporal Sparsity Detection

The temporal sparsity detector calculates the output per-channel sparsity and assigns each channel as either dense or sparse. The sparsity threshold distinguishing dense from sparse channels is determined to balance the execution time between the dense PE and sparse PE. Figure 11 (left) analyzes the sparsity threshold. Based on this analysis, we select 30% as the sparsity threshold. This threshold achieves an average sparsity of 70% for the sparse tensor portion while maintaining a balance between the workloads of the dense and sparse PEs.

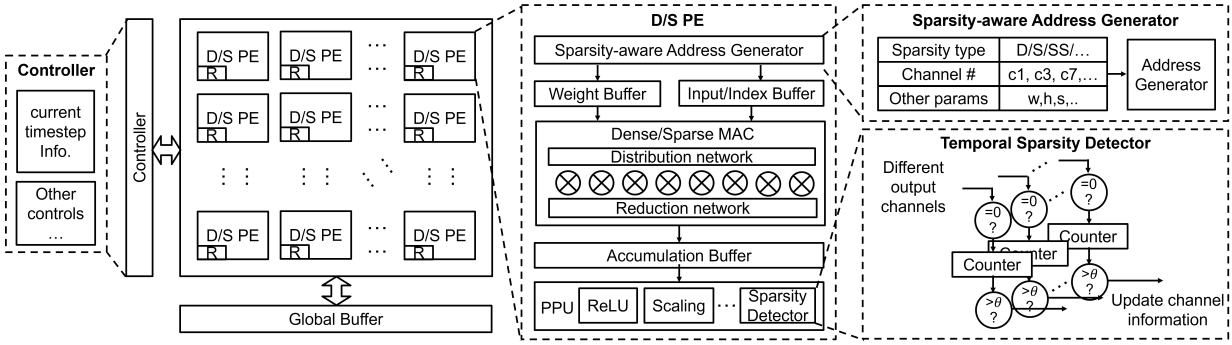


Fig. 9: Overall architecture of our diffusion model accelerator.

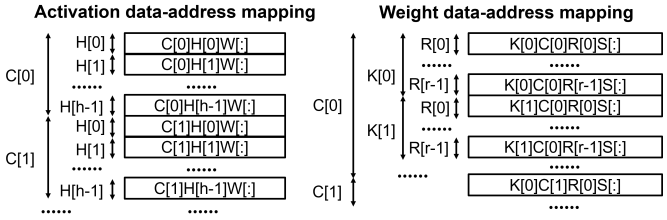


Fig. 10: Channel-last data-address mapping.

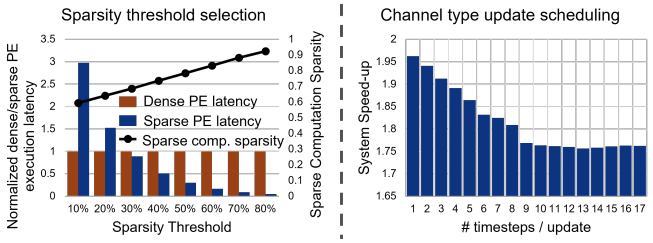


Fig. 11: Analysis for temporal sparsity detection.

Finally, we propose a temporal sparsity update scheduling mechanism to address the dynamic per-channel sparsity observed across time steps. Figure 11 (right) illustrates the relationship between sparsity update frequency and system speed-up. More frequent sparsity updates (i.e., fewer time steps in between updates) improves the accuracy of sparse/dense tensor categorization, resulting in higher system speed-up. Since the overhead of sparsity updates is negligible compared to the overall computation cost and can be hidden behind computation latency, we choose to update the per-channel sparsity at every time step during diffusion model execution to maximize categorization accuracy and system speed-up.

D. Hardware Evaluation

We utilize Stonne [38], an open-source simulation framework, to evaluate the latency and energy consumption of the DPE and SPE. Stonne provides end-to-end evaluation of flexible accelerator microarchitectures with sparsity support. The design is simulated under 28nm technology. In these experiments, we assume the architecture includes one DPE and one SPE, each containing 128 multipliers. This architecture is scalable to meet specific latency and power requirements. The baseline for comparison is a purely dense architecture with two DPEs. Figure 12 (top) presents the average speed-up and energy saving across different datasets relative to the baseline. Focusing solely on temporal sparsity in this figure, we achieve an average speed-up of $1.83\times$ along with a system energy saving of 51.5%.

Figure 12 (bottom) illustrates the total speed-up compared to an FP16 SiLU-based diffusion model. Our 4-bit quantization contributes to $3.78\times$ speed-up, while our temporal per-channel sparsity adds $1.83\times$ speed-up on top of quantization. In combination, our proposed model optimizations and hardware

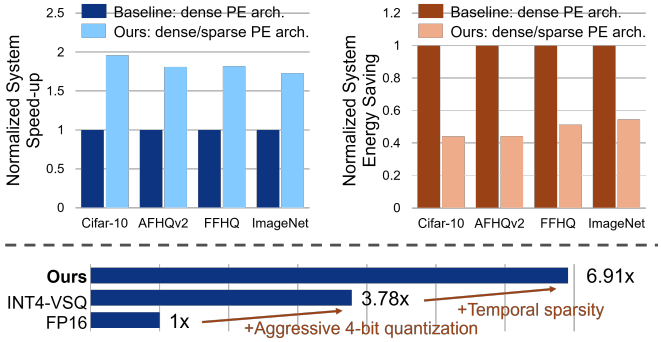


Fig. 12: System evaluation.

architecture described in this paper achieve a total speed-up of $6.91\times$.

V. Conclusions

In this work, we propose a set of co-designed optimization techniques to aggressively quantize diffusion models to 4-bit while simultaneously promoting significant activation sparsity. By designing a novel heterogeneous dense/sparse accelerator architecture, we achieve a $6.91\times$ speed-up compared to an FP16 baseline while demonstrating state-of-the-art image generation quality. By leveraging temporally sparse computations, we save 51.5% in energy consumption compared to traditional dense accelerators. In the future, we plan to extend our techniques to diffusion models targeting video generation [39] and apply our methodology to other generative models. Exploring new DNN models will allow us to further enhance our optimization techniques and expand the applicability of our proposed accelerator design.

REFERENCES

[1] Y. Balaji, S. Nah, X. Huang, A. Vahdat, J. Song, Q. Zhang, K. Kreis, M. Aittala, T. Aila, S. Laine, *et al.*, "eDiff-I: Text-to-Image Diffusion Models with an Ensemble of Expert Denoisers," *arXiv preprint arXiv:2211.01324*, 2022.

[2] A. Blattmann, R. Rombach, H. Ling, T. Dockhorn, S. W. Kim, S. Fidler, and K. Kreis, "Align Your Latents: High-Resolution Video Synthesis with Latent Diffusion Models," *Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[3] M. Mardani, N. Brenowitz, Y. Cohen, J. Pathak, C.-Y. Chen, C.-C. Liu, A. Vahdat, K. Kashinath, J. Kautz, and M. Pritchard, "Generative Residual Diffusion Modeling for Km-scale Atmospheric Downscaling," *arXiv preprint arXiv:2309.15214*, 2023.

[4] T. Karras, M. Aittala, T. Aila, and S. Laine, "Elucidating the Design Space of Diffusion-based Generative Models," *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[5] T. Karras, M. Aittala, J. Lehtinen, J. Hellsten, T. Aila, and S. Laine, "Analyzing and Improving the Training Dynamics of Diffusion Models," *arXiv preprint arXiv:2312.02696*, 2023.

[6] S. Dai, R. Venkatesan, M. Ren, B. Zimmer, W. Dally, and B. Khailany, "VS-Quant: Per-vector Scaled Quantization for Accurate Low-Precision Neural Network Inference," *Conf. on Machine Learning and Systems (MLSys)*, 2021.

[7] Y. Shang, Z. Yuan, B. Xie, B. Wu, and Y. Yan, "Post-Training Quantization on Diffusion Models," *Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[8] X. Li, Y. Liu, L. Lian, H. Yang, Z. Dong, D. Kang, S. Zhang, and K. Keutzer, "Q-Diffusion: Quantizing Diffusion Models," *Int'l Conf. on Computer Vision (ICCV)*, 2023.

[9] Y. He, L. Liu, J. Liu, W. Wu, H. Zhou, and B. Zhuang, "PTQD: Accurate Post-Training Quantization for Diffusion Models," *arXiv preprint arXiv:2305.10657*, 2023.

[10] Y. Yang, X. Dai, J. Wang, P. Zhang, and H. Zhang, "Efficient Quantization Strategies for Latent Diffusion Models," *arXiv preprint arXiv:2312.05431*, 2023.

[11] H. Sun, C. Tang, Z. Wang, Y. Meng, X. Ma, W. Zhu, *et al.*, "TMPQ-DM: Joint Timestep Reduction and Quantization Precision Selection for Efficient Diffusion Models," *arXiv preprint arXiv:2404.09532*, 2024.

[12] Y. Huang, R. Gong, J. Liu, T. Chen, and X. Liu, "TFMQ-DM: Temporal Feature Maintenance Quantization for Diffusion Models," *Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[13] M. Li, Y. Lin, Z. Zhang, T. Cai, X. Li, J. Guo, E. Xie, C. Meng, J.-Y. Zhu, and S. Han, "SVDQunat: Absorbing Outliers by Low-Rank Components for 4-Bit Diffusion Models," *arXiv preprint arXiv:2411.05007*, 2024.

[14] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating Sparse Deep neural Networks," *arXiv preprint arXiv:2104.08378*, 2021.

[15] G. Fang, X. Ma, and X. Wang, "Structural Pruning for Diffusion Models," *arXiv preprint arXiv:2305.10924*, 2023.

[16] K. Wang, J. Chen, H. Li, Z. Mi, and J. Zhu, "SparseDM: Toward Sparse Efficient Diffusion Models," *arXiv preprint arXiv:2404.10445*, 2024.

[17] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," tech. rep., University of Toronto, 2009.

[18] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha, "StarGAN v2: Diverse Image Synthesis for Multiple Domains," *Conf. on Computer Vision and Pattern Recognition*, 2020.

[19] T. Karras, S. Laine, and T. Aila, "A Style-based Generator Architecture for Generative Adversarial Networks," *Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," *Conf. on Computer Vision and Pattern Recognition*, 2009.

[21] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[22] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, *et al.*, "Microscaling Data Formats for Deep Learning," *arXiv preprint arXiv:2310.10537*, 2023.

[23] A. Tirumala and R. Wong, "NVIDIA Blackwell Platform: Advancing Generative AI and Accelerated Computing," *IEEE Hot Chips Symp. (HCS)*, 2024.

[24] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning," *Neural Networks*, 2018.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[26] C. Shinn, C. McCarthy, S. Muralidharan, M. Osama, and J. D. Owens, "The Sparsity Roofline: Understanding the Hardware Limits of Sparse Neural Networks," *arXiv preprint arXiv:2310.00496*, 2023.

[27] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, *et al.*, "MAGNet: A Modular Accelerator Generator for Neural Networks," *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2019.

[28] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter Performance Analysis of a Tensor Processing Unit," *Int'l Symp. on Computer Architecture (ISCA)*, 2017.

[29] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[30] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," *ACM SIGARCH Computer Architecture News*, 2016.

[31] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An Accelerator for Sparse Neural Networks," *Int'l Symp. on Microarchitecture (MICRO)*, 2016.

[32] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks," *Int'l Symp. on Microarchitecture (MICRO)*, 2019.

[33] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "ExTensor: An Accelerator for Sparse Tensor Algebra," *Int'l Symp. on Microarchitecture (MICRO)*, 2019.

[34] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training," *Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2020.

[35] N. Srivastava, H. Jin, S. Smith, H. Rong, D. Albonesi, and Z. Zhang, "Tensaurus: A Versatile Accelerator for Mxed Sparse-Dense Tensor Computations," *Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2020.

[36] J. H. Shin, A. Shafiee, A. Pedram, H. Abdel-Aziz, L. Li, and J. Hassoun, "Griffin: Rethinking Sparse Optimization for Deep Learning Architectures," *Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2022.

[37] E. Qin, R. Garg, A. Bambhaniya, M. Pellauer, A. Parashar, S. Rajamanickam, C. Hao, and T. Krishna, "Enabling Flexibility for Sparse Tensor Acceleration via Heterogeneity," *arXiv preprint arXiv:2201.08916*, 2022.

[38] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "STONNE: Enabling Cycle-Level Microarchitectural Simulation for DNN Inference Accelerators," *Int'l Symp. on Workload Characterization (IISWC)*, 2021.

[39] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video Diffusion Models," *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

XShift: FPGA-efficient Binarized LLM with Joint Quantization and Sparsification

Shuai Zhou, Huinan Tian, Sisi Meng, Jianli Chen, Jun Yu, and Kun Wang†
State Key Lab of Integrated Chips & Systems, and School of Microelectronics, Fudan University, Shanghai, China
†Corresponding author: kun.wang@ieee.org

***Abstract*—Binarization is a promising approach to significantly reduce computational complexity by replacing multiplications with hardware-efficient XNOR operations. However, the binarization of LLM activations often leads to severe accuracy degradation, while weight-only binarization fails to eliminate multipliers due to the Self-Attention mechanism. Furthermore, LLMs exhibit distinctive channel-level data distribution characteristics and differing computational and memory requirements between the Pre-fill and Decoding stages, necessitating a specialized inference framework.**

In response, we introduce *XShift*, an algorithm-hardware co-design framework optimized for efficient binarized LLM inference on FPGAs. *XShift* incorporates three key contributions: (1) a hardware-friendly XNOR-Shift Encoding (XSE) format that transforms traditional multiplications into XNOR and shift operations, ensuring scalability and precision; (2) Hardware Adaptive Outlier and Sparsity (HAOS) techniques, which exploit channel-level data distribution and systolic array architectures for optimized quantization and sparsification; and (3) a dedicated hardware accelerator featuring an XNOR-Shift Systolic Array (XSSA) and an enhanced Base-2 SoftMax Converter (BSMC), designed to address the specific computational demands of binarized LLMs. Experimental evaluations on the Alveo U280 and U50 FPGA demonstrate that *XShift* achieves a 10-15× reduction in DSP resource usage while surpassing existing accelerators and GPUs in inference performance. Specifically, *XShift* delivers an average speedup of 4.17-4.76× and a 14.29-6.95× improvement in energy efficiency, alongside lower perplexity compared to other low-precision LLM techniques. These results underscore the potential of *XShift* for edge deployment of LLMs.

I. Introduction

The evolution of Large Language Models (LLMs) since the introduction of GPT-1 [1] in 2018, which contained approximately 0.1 billion parameters, has been extraordinary. Within just six years, SOTA models like LLaMA-3 [2] now encompass up to 405 billion parameters. Utilizing BF16 precision, such massive models require at least 810 GB of GPU memory, typically distributed across two systems equipped with 8 Nvidia H100 GPUs each. This exponential growth in size and computational demands presents substantial challenges for deployment, particularly in resource-constrained environments.

To mitigate these challenges, various model compression techniques have been developed, including quantization [3]–[11] and sparsification [12]–[14]. Additionally, hardware accelerators based on FPGA [15]–[17] and ASIC [18]–[22] have been explored. For example, FlightLLM [15] integrates mixed precision and sparsity into an accelerator, but its additional hardware overhead and limited compatibility with systolic arrays constrain its flexibility. Moreover, its inability to quantize activations below 8 bits remains a bottleneck.

Binarization, a more hardware-friendly alternative to low-precision quantization, has emerged as a promising technique [3], [4], [23], [24]. However, current efforts in binarizing Transformers have been limited to models with relatively few parameters, such as BERT [23] and ViT [24]. While weight binarization has been partially achieved for larger LLMs [3], [4], it fails to eliminate attention matrix multiplications during inference. Furthermore, high-precision activations, particularly in the decoding stage where KV cache is prominent, substantially increase hardware storage demands. Fully binarizing both weights and activations often results in severe accuracy degradation, and the lack of specialized hardware support has hindered its practical application.

In addition, LLMs exhibit distinct computational and memory characteristics between the Pre-fill and Decoding stages [25], [26]. In edge computing scenarios, where resources are limited, achieving efficiency is critical. Users expect minimal latency during the Pre-fill stage and token output rates during the Decoding stage that align with natural reading speeds of 150–300 words per minute [27]. Designing inference frameworks that cater to these requirements while optimizing hardware utilization remains a significant challenge.

In this paper, we propose **XShift**, an algorithm-hardware co-design framework tailored for efficient binarized LLM inference on FPGAs. The key contributions of our work are summarized as follows:

- We introduce a hardware-efficient XNOR-Shift Encoding (XSE) format that eliminates the need for DSPs in matrix computations. The XSE format supports both standard low-bit quantization and an extended version for higher representational range.
- We propose Hardware Adaptive Outlier and Sparsity (HAOS) techniques that leverage LLM-specific channel-level data distributions and systolic array architectures to achieve highly compressed yet accurate models.
- We develop a dedicated hardware accelerator featuring an XNOR-Shift Systolic Array (XSSA) with three computation modes, optimized for LLM inference. Additionally, we design a Base-2 SoftMax Converter (BSMC) for efficient SoftMax processing on FPGA.
- Through extensive experimentation, *XShift* achieves higher accuracy compared to other low-precision LLM approaches. It also outperforms leading accelerators and GPUs in terms of inference speed and energy efficiency.

II. Background and Motivation

A. Large Language Models

Traditional metrics such as end-to-end (E2E) latency and throughput are insufficient for evaluating the inference performance of Large Language Models (LLMs) due to the distinct characteristics of the Pre-fill and Decoding stages. Instead, metrics like Time to First Token (TTFT) and Time Between Tokens (TBT) are more appropriate [26]. As illustrated in Fig. 1(a), TTFT measures the latency from the reception of a user request to the generation of the first output token, where lower values are critical for a responsive user experience. TBT quantifies the interval between the generation of consecutive tokens during the Decoding stage, aligning with typical reading speeds.

In real-world edge deployment scenarios, user requests are sporadic and often unbatchable, and the number of iterations in the

This work was financially supported in part by the National Key Research and Development Program of China under Grant 2021YFA1003602, in part by Project of Shanghai EC under Grant 24KXZNA12, in part by National Natural Science Foundation of China (NSFC) Research Project 92373207. The computations in this research were performed using the CFFF platform of Fudan University.

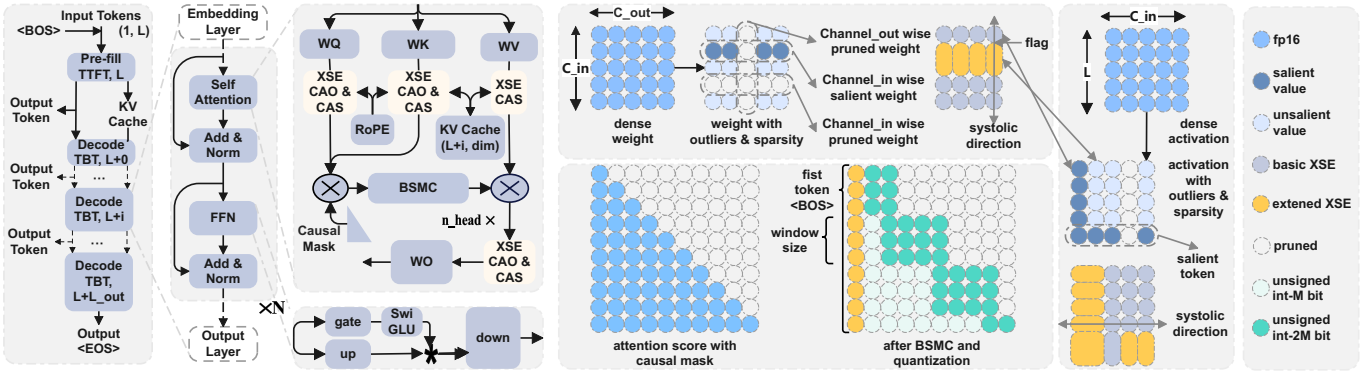


Fig. 1. Network architecture of the proposed *XShift*: (a) LLM inference with one Pre-fill stage and iterative Decoding stage; (b) Detailed attention mechanism of our *XShift* model; (c) Weights and activations optimized using HAOS techniques.

Decoding stage varies significantly across requests. Unlike data center environments, where batching multiple user requests optimizes GPU utilization, edge devices frequently encounter discontinuous requests [25]. Batching under these conditions can introduce unnecessary delays, as new requests need to wait for ongoing Decoding processes to complete. Moreover, even if different inputs are batched together, the variability in Decoding iteration counts across requests can result in hardware underutilization, reducing overall inference efficiency.

As depicted in Fig. 2, an effective edge inference framework must accommodate three operational modes. During the Pre-fill stage, the framework should minimize the average TTFT for each user request by optimizing weight reuse through fine-grained batching. In the Decoding stage, the framework must balance energy efficiency with maintaining a TBT speed suitable for natural reading rates.

B. LLM Compression Techniques

Quantization of both activations and weights is a widely adopted approach to enhance hardware efficiency. However, it poses significant challenges, particularly in addressing outliers that deviate markedly from the typical data distribution. Techniques as [8], [28], [29] manage outliers by isolating them into sparse matrices for specialized processing. Methods like [20], [22] employ paired strategies to reduce the overhead of managing outliers separately, while OPAL [21] uses mixed precision and specialized formats to manage a fixed number of outliers per block. Channel-level observations have also inspired approaches [6], [7], [10], [30], which employ channel-wise scaling to shift quantization difficulty from activations to weights.

Sparsity in LLMs is another promising avenue for compression. [12], [14] utilize unstructured sparsity to minimize performance degradation during compression. Meanwhile, structured pruning approaches [13], [31] focus on improving hardware compatibility. Joint sparsity and quantization techniques, as exemplified by JSQ [32], have shown significant promise in achieving high compression rates with minimal accuracy loss.

Among all compression methods, binarization [3], [4], [23], [24] stands out for its superior compression ratios and its ability to eliminate multiplications, replacing them with more efficient operations. While Binary Neural Network (BNN)-specific accelerators [33], [34] have demonstrated potential, they often lack support for LLM architectures, underscoring the need for tailored hardware designs that fully exploit the advantages of LLM binarization.

C. LLM Deployment on Edge Devices

Complementing LLM algorithmic optimizations, various LLM accelerators based on [15]–[17] and ASIC [18]–[22] have been proposed to enhance deployment efficiency. Previous works have focused on designing accelerators to speed up inference in Transformer-based neural networks [18]. Some exploit model sparsity by designing specialized hardware modules to accelerate computations involving unstructured or structured sparsity. Others focus on quantization; for example, FACT [18] demonstrates the benefits of mixed-precision quantization in improving Transformer inference speed.

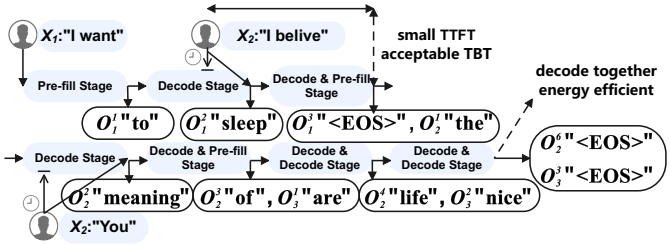


Fig. 2. Illustration of *XShift*'s effectiveness in handling discontinuous requests using the XSSA with three computation modes.

Unlike earlier works that primarily targeted BERT or ViT models, recent research has shifted toward accelerating larger LLMs. Projects like OPAL [21], OliVe [20], and Oltron [22] adopt specialized quantization schemes to accommodate the increased complexity. FlightLLM [15] designs an accelerator supporting both mixed precision and sparsity in matrix multiplication, but this introduces additional hardware overhead and may not integrate well with systolic arrays.

III. XNOR-Shift Encoding (XSE)

A. Basic XSE Format Principles

In this section, we present the XNOR-Shift Encoding (XSE) format, a simple yet effective method that provides customizable bit-width configurations for enhanced accuracy while preserving the DSP-free characteristics of Binarized Neural Networks (BNNs) for matrix computation. As depicted in Fig. 3(a) (bottom), shift operations are approximately 8× more resource and energy efficient than multiplication and addition units in low-bit computations, making the XSE format highly suitable for hardware-optimized low-bit quantization in LLMs.

The XSE format enables computational efficiency by replacing multiplications with XNOR and shift operations. As shown in Fig. 3(a) (top), the 3-bit XSE format encodes two data values using a straightforward base-2 scheme. This approach transforms multiplication into an XNOR operation applied to the sign bit and a shift operation applied to the shift bit, significantly reducing computational overhead. Furthermore, this basic framework can be

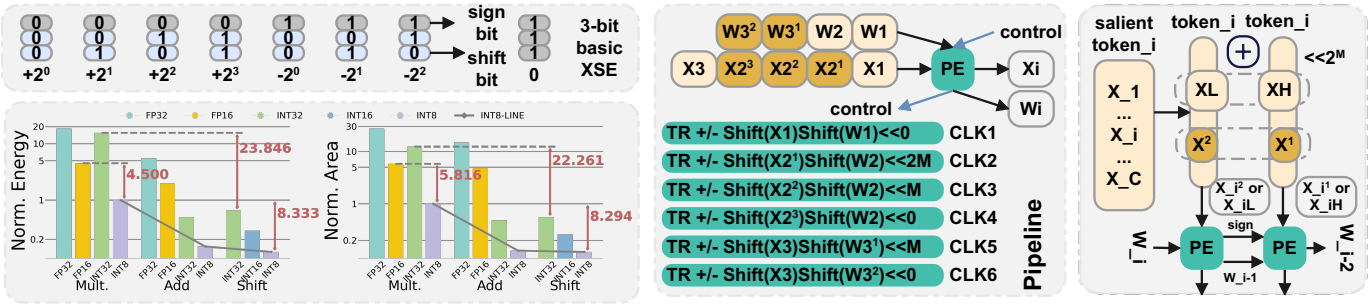


Fig. 3. Principles of XSE: (a) Basic 3-bit XSE format, along with energy and area comparisons between multiplication, addition, and shift operations; (b) Computation flow for channel-wise extended XSE; (c) Calculation principle for salient token with extended XSE.

extended to higher bit-width configurations, providing a broader range and improved precision.

B. Extended XSE for Outliers

Traditional methods that treat outliers as sparsity, often introduce significant overhead in hardware resources and runtime bandwidth. To address these limitations, we propose an extended XSE format that balances computation time and hardware efficiency. This format aligns seamlessly with the bandwidth requirements of the basic XSE format and integrates effectively with systolic array computations. With minimal hardware costs, the extended XSE format enables computations to be distributed across multiple clock cycles.

$$(-1)^{\text{sign}} \times (1 << 2N + I_H << N + I_L << 0) << \text{shift}.$$

The extended $3 \times N$-bit XSE format adopts a floating-point-like representation. This scalable-width mantissa configuration provides a broader range. Extending an N-bit XSE format to an $M \times N$-bit XSE format involves designating $(M-1) \times N$ bits as an unsigned Int, $N-1$ bits for the exponent shift, and one bit for the sign. This enhancement significantly improves the precision of outliers, effectively capturing critical data that influence model accuracy.

The computation flow for multiplying extended $M \times N$-bit XSE data $X2$ with basic N-bit XSE data $W2$ is illustrated in Fig. 3(b), where the extended XSE format is applied to the channel-wise of activations: (1) The 1-bit sign and 2-bit shift of $X2^1$ and $W2$ are introduced. A flag shifts the data by six bits before it enters the adder, partially computing the product. (2) The upper 3-bit mantissa of $X2^2$ is transmitted, applying an additional 3-bit shift to compute the second component while $W2$ remains queued for the next cycle. (3) The lower 3-bit mantissa of $X2^3$ is processed, completing the computation, while $W2$ continues to propagate through the pipeline.

As the pipeline advances, new data ($X2$ and $W2$) are sequentially fed into the computation units, enabling efficient and continuous processing. The extended XSE format significantly broadens the applicability of the basic XSE scheme, enabling precise handling of outliers in large models. It achieves high accuracy without necessitating substantial hardware modifications. Furthermore, the XSE format remains DSP-free and straightforward to extend, making it an ideal choice for enhancing the accuracy of BNNs.

IV. Hardware Adaptive Outlier and Sparsity

Using only magnitude as a metric is insufficient, as it neglects their cumulative impact on the model's output. Inspired by Optimal Brain Damage (OBD) [35], previous methods for outlier quantization [8] and sparsification [12], [14], [17] have used a sensitivity matrix to measure errors, formulated as:

$$\begin{aligned}\mathbb{S}_W &= (W - \mathbb{Q}(W))^2 / \text{diag}[(2X^T X + \lambda I)^{-1}] \\ &\Rightarrow (W - \mathbb{Q}(W))^2 \parallel X_{:,j} \parallel_2^2 .\end{aligned}$$

$(2X^T X + \lambda I)^{-1}$ is the Hessian H, and λ is a dampening factor to prevent instability in the inverse computation. When $\lambda = 0$ and only the diagonal elements of H are retained, the $O(d^3)$ complexity of H^{-1} simplifies to the L2 norm of X with $O(d^2)$ complexity.

Activations X, often exhibiting high magnitudes at the channel level, are adjusted using a channel-wise bias term for activations:

$$\begin{aligned}XW &= (\hat{X} \odot S_{(1,C_{in})} + B_{(1,C_{in})}) \cdot (\hat{W} \oslash S_{(1,C_{in})}) \\ &= \hat{X} \times \hat{W} + \hat{B}_{(1,C_{out})}, \\ \mathbb{S}_W &= (\hat{W} - \mathbb{Q}(\hat{W}))^2 \parallel \hat{X}_{:,j} \parallel_2^2, \\ \mathbb{S}_X &= (\hat{X} - \mathbb{Q}(\hat{X}))^2 \parallel \hat{W}_{i,:} \parallel_2^2,\end{aligned}$$

where the channel-wise bias B represents the average channel value of X with dimensions (L, C_{in}), while $\hat{B} = B \cdot W$ ensures that bias adjustments without affect final outputs. Inspired by [5]–[7], [10], channel-wise scaling S redistributes quantization difficulty. To compensate for irregular weight W distributions, $W + ab^T$ is used, where $W \in \mathbb{R}^{C_{in} \times C_{out}}$, $a \in \mathbb{R}^{C_{in}}$, and $b \in \mathbb{R}^{C_{out}}$. $\mathbb{Q}(\cdot)$ uses the XSE format in Sec. III with additional learnable parameters α and β to control clipping ranges. During sparsity sensitivity, quantized values $\mathbb{Q}(\hat{X})$ and $\mathbb{Q}(\hat{W})$ are treated as zero. The required additional parameters are minimal and learnable, trained using a small calibration dataset. For activation sensitivity $\mathbb{S}_X$, we identify outlier channels using a metric that considers quantization error rather than solely magnitude or variance (e.g., the 3-σ rule in [20], [22]). This approach accounts for the influence of weights on activations, crucial for the XSE format. Unlike $\mathbb{S}_W$, which can be used to update weights [8], [14], [36], $\mathbb{S}_X$ assesses activation sensitivity to determine outlier channels only, as activations cannot be updated.

Considering *XShift*'s hardware constraints, outlier detection and pruning occur statically at the channel level, avoiding outliers multiplication within the same C_{in}. Quantify and inference separately for activation and weight on the conflicting outlier dimension to determine which one has a greater impact on the output result, in order to decide which can preserve this outlier channel. By organizing computations along the channel dimension, stepwise propagation is maintained, ensuring compatibility with the systolic array architecture. For both activations and weights, specific channels will increased bit widths, while less significant channels are pruned.

Additionally, specific outlier token support ensures accuracy for tokens requiring greater attention. Special tokens like $<$ BOS $>$ and

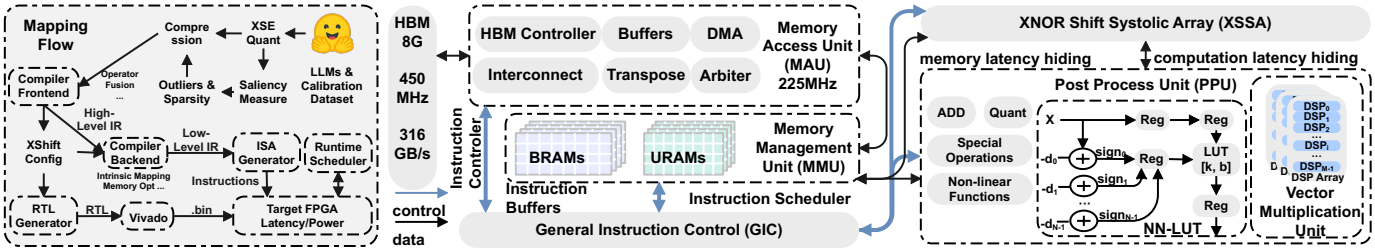


Fig. 4. Comprehensive architecture of *XShift*: (a) General mapping flow; (b) Overview of hardware architecture; (c) Details of the Post-Process Unit (PPU) featuring NN-LUT and a shared Vector Multiplication Unit.

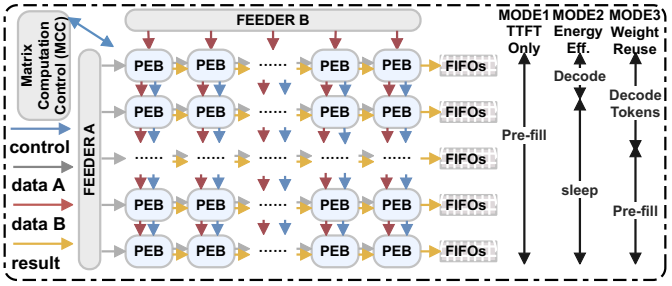


Fig. 5. Detailed architecture of XNOR Shift Systolic Array (XSSA) with support for all three computation modes.

$<$ CLS $>$, which contain global information critical for tasks in language generation and classification [37]. Fig. 3(c) illustrates the principle of expanding one token into M tokens with extended XSE activation values. This process maintains systolic array efficiency by passing XNOR results to subsequent PEs and summing M tokens to produce the outlier token result. For outlier weights, corresponding activation channels use basic XSE to avoid calculation conflicts. *XShift* supports N outlier tokens per K tokens depending on the systolic array size in FEEDER A.

V. Hardware Architecture

A. XShift Mapping Flow

As illustrated in Fig. 4(a), the algorithmic optimizations detailed in Sec. III and Sec. IV enable the creation of a highly compressed model with superior accuracy. Once the model is optimized, it is processed through a compiler to generate the network graph pattern in High-Level Intermediate Representation (IR). This representation is then passed to the compiler backend for hardware-specific optimizations. The final Low-Level IR is subsequently translated by the ISA Generator into optimized instructions and model parameters, tailored for hardware deployment.

Recognizing the unique requirements of the Pre-fill and Decoding stages in LLMs, as well as the intermittent nature of user requests in edge applications, the *XShift* hardware design incorporates a parameter-sharing-driven scheduler and a versatile XNOR-Shift Systolic Array (XSSA), as depicted in Fig. 5. This architecture is capable of supporting three distinct computation modes, each designed to optimize efficiency and responsiveness.

When a user initiates a request, the system begins with the Pre-fill stage. During this phase, the scheduler activates the first XSSA computation mode, where all Processing Elements (PEs) within the XSSA are utilized to accelerate the process and minimize latency. Any additional user requests are queued during this time. Upon completion of the Pre-fill stage for the initial request, the system transitions to the Decoding stage, which demands less computational power and latency. In this second computation mode, only the first layer of the XSSA is active, while other layers are idled to conserve static power. The processing frequency in this mode can be dynamically adjusted to meet latency requirements. For instance, maintaining a 200 ms interval per Decoding stage iteration aligns with typical English reading speeds of 300 words per minute.

If a new request arrives while a Decoding stage is in progress, the standard approach would require the new request to wait until the current Decoding stage completes. However, Decoding stage latency can vary significantly, often lasting several seconds depending on the token count, making this approach impractical. The *XShift* design mitigates this issue by allowing the Pre-fill stage of the new request to batch with the ongoing Decoding stage, thereby sharing weight parameters to reduce latency. Furthermore, multiple Decoding stages can be batched together for matrix multiplications involving the weight matrix. In this third computation mode, the scheduler ensures the ongoing Decoding maintains its token-per-minute output while maximizing parameter reuse during activation-weight multiplications. This strategy minimizes the wait time for new Pre-fill and ongoing Decoding stages, significantly improving overall system efficiency and responsiveness.

B. XNOR-Shift Systolic Array

To efficiently execute the matrix computations required by the *XShift* model, we introduce a 2-D XNOR-Shift Systolic Array (XSSA), meticulously designed to achieve both computational efficiency and scheduling requirements. The XSSA architecture is organized into K layers, each comprising M Processing Element Blocks (PEBs). Each PEB contains $n1 \times n2$ Processing Elements (PEs), resulting in a peak processing capacity of $K \times M \times n1 \times n2$ Multiply-Accumulate (MAC) operations per clock cycle. This systolic array structure optimizes data reuse and minimizes bandwidth demands, making it highly effective for parallel matrix multiplications.

Data is fed into the systolic array through two input feeders, FEEDER A and FEEDER B. FEEDER A provides input data at a bandwidth of $D_A \times K$ bits, while FEEDER B operates at $D_B \times M \times n1 \times n2$ bits, propagating data horizontally and vertically through the array. The Matrix Computation Control (MCC) unit oversees the pipeline operations, managing process initiation, pauses, and completion while also handling control signals. When a channel contains an outlier, the MCC dispatches specific flag to the PEs, with commands cascading through the array. Similarly, result timing follows a sequential pattern, with outputs being transmitted to the FIFO buffer at a bandwidth of $D_O \times K \times n1 \times n2$ bits. These outputs can then undergo dequantization or other forms of post-processing.

The design of the PEB strategically minimizes the systolic array depth M, which reduces waiting times and lowers intermediate register costs. Each PEB encompasses $n1 \times n2$ PEs that function simultaneously on the same clock cycle. As shown in Fig. 6, the PEs

Algorithm 1: Base-2 SoftMax Converter (BSMC) algorithm in *XShift*

Input : Result after QK^T as $\{a_0, ..., a_{N-1}\}$ with a mask size of N, Scale value pre-compute $\omega_0 \times log_2(e)$ and $\omega \times log_2(e)$
Output: Quantified SoftMax output $\{s_0, ..., s_{N-1}\}$, Scale value α

$\alpha \times \{s_0, ..., s_{N-1}\} = \frac{exp(\hat{\omega} \times a_i)}{exp(\omega_0 \times a_0) + \sum_1^{N-1} exp(\omega \times a_i)}$;
where $\hat{\omega} \leftarrow \omega_0, if: i = 0; \hat{\omega} \leftarrow \omega, otherwise$;
Initialize $I_{max} = 0, SUM = 0$;
for $i \leftarrow 0$ **to** $N-1$ **do**
  $x_i \leftarrow a_i \times (\hat{\omega} \times log_2(e))$; /* Convert $e^{a_i \times \hat{\omega}}$ to 2^{x_i} */
  $I_i, d_i \leftarrow x_i$; /* I_i is the integer part, d_i is the decimal part */
  $I_{max} = max(I_{max}, I_i), i \neq 0$; $k_i, b_i \leftarrow d_i$; /* Approximate $2^{d_i} = d_i \times k_i + b_i$ */
for $i \leftarrow 0$ **to** $N-1$ **do**
  $s_i \leftarrow (k_i \times d_i + b_i) >> (I_{max} - I_i); SUM = SUM + s_i$;
  $s_0 \leftarrow XSE_{extend}(s_0), if: i = 0; s_i \leftarrow UINT(s_i, 2M), if: i \in window; s_i \leftarrow UINT(s_i, M), otherwise$;
$\alpha = 1/SUM$;
return $\{s_0, ..., s_{N-1}\}$, α;

are designed with simplicity and efficiency. During each cycle, the PEs receive input data A and B, apply an XNOR operation on their sign bits, and execute bit-shift operations on the remaining bits. The resulting values are then combined through addition or subtraction, determined by the XNOR outcome. If the outlier control signal flags A as an outlier, additional bits are introduced in subsequent cycles, with corresponding bit-shift operations and accumulations conducted as necessary. This approach minimizes hardware adjustments while ensuring efficient processing. The processed data, results, and control signals are then propagated to subsequent stages for reuse, maintaining a seamless flow of computations.

C. Base-2 SoftMax Converter

The SoftMax function, a critical component in LLM post-processing, presents significant challenges due to its high computational complexity and stringent precision requirements, particularly when computing exponentials such as $e^{s \cdot A}$. Without simplifications, these calculations demand substantial resources and result in a wide dynamic range of outputs, exacerbating errors from piecewise linear approximations more than other functions. Additionally, the strictly positive nature of SoftMax outputs and the variable input lengths in LLMs necessitate dynamic quantization to prevent computational losses in subsequent stages.

To address these challenges, we propose a hardware-efficient solution: the Base-2 SoftMax Converter (BSMC). This approach achieves both accuracy and efficiency in SoftMax calculations without requiring model retraining. The BSMC method, outlined in Algorithm 1, begins by scaling the XSSA output, a_i, by a factor ω, to compute $e^{s \cdot A}$. Instead of directly calculating $e^{\omega \cdot a_i}$, ω is replaced by $\omega \cdot \log_2(e)$, transforming $e^{\omega \cdot a_i}$ into $2^{\omega \cdot \log_2(e) \cdot a_i}$. This precomputation of $\omega \cdot \log_2(e)$ significantly improves computational efficiency.

It is worth noting that the first token, represented as $< BOS >$, is assigned its own scaling factor, ω_0, and is quantified using an extended XSE format. This token is critical for downstream computations and has a significant impact on overall results. For each element a_i, we calculate $\omega \cdot \log_2(e) \cdot a_i$, decomposing it into an integer part I_i and a decimal part d_i. During this process, $I_{\max}$, the maximum integer value across all I_i (excluding the first token), is identified. The expression $2^{\omega \cdot \log_2(e) \cdot a_i}$ is then simplified to $2^{d_i} << I_i$, where $d_i \in [0, 1)$ can be efficiently approximated using NN-LUT [38]. The NN-LUT employs parameters k_i and b_i for piecewise linear interpolation, approximating 2^{d_i} as $k_i \cdot d_i + b_i$. This restricted range for d_i ensures high precision.

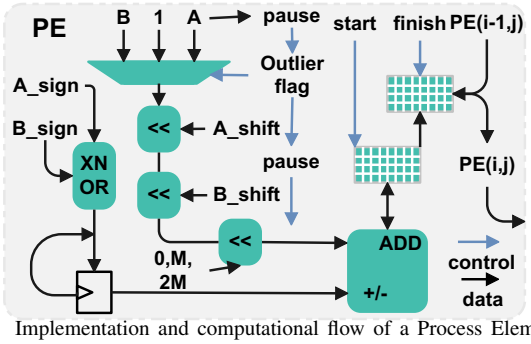


Fig. 6. Implementation and computational flow of a Process Element (PE) within the XSSA.

In subsequent quantization, the exponent is adjusted by subtracting $I_{\max}$, preserving the SoftMax computation while simplifying each term s_i as $(k_i \cdot d_i + b_i) >> (I_{\max} - I_i)$. The cumulative sum, SUM, is computed concurrently. Instead of directly normalizing s_i by multiplying with $\frac{1}{\text{SUM}}$, we leverage the earlier adjustment to quantize s_i as an unsigned integer. This eliminates the need for a sign bit, extends the range of representable values, and allows for higher precision within a predefined window. The scaling factor $\frac{1}{\text{SUM}}$ is then designated as row-specific and seamlessly integrates with subsequent matrix computations involving V. Importantly, the transposed V matrix remains free of outlier quantization, ensuring full compatibility with the XSSA matrix multiplication framework.

VI. Experimental Results

Models and Datasets. The *XShift* framework employs advanced compression techniques from Sections III and IV to optimize LLaMA family models [2], including LLaMA, LLaMA-2, and the latest LLaMA-3. We evaluate perplexity (PPL) as the primary metric for language generation tasks, using the WikiText2 [39] and C4 [40] datasets. Additionally, zero-shot task performance is evaluated via the lmevaluation-harness framework [41], which covers a range of tasks including PiQA, ARC, BoolQ, and HellaSwag, to assess the generalization capabilities of *XShift*.

Experimental Settings. For software-based evaluations, CUDA Toolkit 11.8 and PyTorch 2.0.0 were used. Calibration involved 128 samples randomly extracted from WikiText2, each consisting of 2048 tokens. Models were fine-tuned over 10–40 epochs, targeting optimal sparsity and quantization levels through a structured pipeline. For hardware-based evaluations, *XShift* is deployed on the Xilinx Alveo U50 and U280 FPGAs. The U50 FPGA, a cost-effective alternative to the U280 ($5000), achieves reduced resource consumption while maintaining high efficiency. Hardware implementations were written in SystemVerilog and operated at 225 MHz for optimal memory bandwidth. The synthesis is conducted using Xilinx Vitis v2023.1 and Vivado toolchains. Comparisons were drawn against Nvidia V100 and A100 GPUs using PyTorch-based implementations (GPU-native) and SmoothQuant-optimized models (GPU-opt).

A. Accuracy Analysis

Quantized Model Performance: The performance of *XShift* is first assessed on WikiText2 and C4 datasets. Table I presents the inference operations (OPs) and perplexity (PPL) results. Compared to state-of-the-art methods such as BiLLM [3] (binarization), Wanda [12] (sparsity), I-LLM [9] (low-precision quantization), Tender [19] and Oltron [25] (software-hardware co-design), *XShift* demonstrates the lowest PPL across both datasets.

TABLE I

NUMBER OF INFERENCE OPERATIONS (OPS) AND PERPLEXITY RESULTS (PPL) OF QUANTIZED MODEL ON (WIKITEXT2 / C4) DATASETS (LOWER IS BETTER).

Method	FP16		BiLLM [3]		Wanda [12]		I-LLM [9]		Tender [19]		Oltron [22]		**XShift$_4$**		**XShift$_3$**	
Models	OPs↓	PPL↓	OPs↓	PPL↓	OPs↓	PPL↓	OPs↓	PPL↓	OPs↓	PPL↓	OPs↓	PPL↓	**OPs↓**	**PPL↓**	**OPs↓**	**PPL↓**
LLaMA-7B	14.64T	5.68/7.08	2.03T	35.04/-	7.87T	8.57/-	0.92T	9.10/12.33	0.92T	23.85/38.09	0.94T	11.67/15.21	**0.84T**	**8.11/11.65**	**0.59T**	**8.42/12.44**
LLaMA-13B	28.05T	5.09/6.61	3.52T	15.14/-	14.89T	7.40/-	1.76T	7.99/10.96	1.76T	13.68/28.24	1.80T	8.20/10.84	**1.58T**	**7.45/9.36**	**1.11T**	**7.78/9.44**
LLaMA2-7B	14.64T	5.47/6.97	2.02T	23.48/-	7.87T	7.97/-	0.92T	10.44/12.92	0.92T	36.47/114.44	-	-/-	**0.76T**	**8.01/10.64**	**0.45T**	**8.32/11.41**
LLaMA2-13B	28.05T	4.88/6.46	3.51T	16.77/-	14.89T	6.55/-	1.76T	9.76/12.57	1.76T	55.08/208.76	-	-/-	**1.44T**	**8.22/10.31**	**0.85T**	**9.65/11.88**
LLaMA3-8B	16.48T	6.14/8.88	2.16T	28.3/2.9E2	-	-/-	1.04T	21.19/30.9	-	-/-	-	-/-	**0.93T**	**14.42/18.45**	**0.66T**	**15.54/20.75**

TABLE II

ACCURACY RESULT ON VARIOUS ZERO-SHOT TASKS UNDER 4-BIT WEIGHT AND ACTIVATION.

Model	**Method**	**PiQA**	**ARC-e**	**ARC-c**	**BoolQ**	**HellaSwag**	**Winogrande**	**Avg.**
LLaMA-13B	FP16	79.10	59.89	44.45	68.01	76.21	70.31	66.32
	OmniQuant [6]	69.69	47.30	33.10	62.84	58.96	55.80	54.62
	I-LLM [9]	67.95	48.15	34.47	62.29	63.13	59.98	55.99
	XShift	**73.88**	**52.40**	**40.96**	**66.44**	**69.40**	**65.23**	**61.39**
LLaMA2-13B	FP16	79.05	57.91	44.19	69.02	76.60	69.69	66.07
	OmniQuant [6]	67.08	45.66	32.25	63.73	58.39	54.61	53.62
	I-LLM [9]	68.00	45.74	30.97	64.55	60.62	54.22	54.01
	XShift	**73.47**	**51.21**	**39.45**	**66.76**	**68.23**	**62.77**	**60.32**

TABLE III

COMPREHENSIVE RESOURCE UTILIZATION COMPARISON OF VARIOUS FPGA-BASED DESIGNS.

Method	**Platform**	**LUT**	**FF**	**BRAM**	**URAM**	**DSP**
DFX [16]	Alveo U280	520K	1107K	1192	104	3533
FlightLLM [15]	Alveo U280	574K	943K	1252	792	6345
XShift	**Alveo U280**	**654K**	**820K**	**935**	**662**	**356**
	Alveo U50	**430K**	**512K**	**564**	**448**	**224**

For instance, on smaller models, *XShift* outperforms Oltron significantly, achieving up to 3 lower PPL. Against BiLLM, *XShift* achieves nearly 50% lower PPL while maintaining fewer OPs, highlighting the effectiveness of its low-precision quantization for both weights and activations. The 4-bit XShift$_4$ variant operates at under 1T OPs for 2048-token inputs, making it ideal for edge deployments, while the compact 3-bit XShift$_3$ variant reduces computational complexity to approximately 0.5T OPs, maintaining competitive PPL performance.

Challenging LLaMA-3 Model: The LLaMA-3 model, known for its quantization difficulties, also highlights *XShift*'s robustness. Compared to I-LLM, *XShift* achieves up to 5 fewer PPL, maintaining similar OP counts. This underscores its ability to balance efficiency and accuracy even under challenging conditions.

Zero-Shot Task Evaluation: Table II highlights zero-shot task performance on LLaMA2-13B under 4-bit weight and activation settings. *XShift* consistently achieves the highest accuracy across tasks, demonstrating a 6.3% improvement in average accuracy on LLaMA-13B and LLaMA2-13B models with superior generalization capabilities. These improvements are attributed to *XShift*'s ability to identify and preserve critical outliers, ensuring robust performance across diverse tasks.

B. Hardware Performance and Utilization

Resource Utilization: Table III compares resource usage across FPGA-based designs, including DFX [16] and FlightLLM [15]. *XShift* achieves the lowest resource utilization, requiring only 224 DSPs on the U50 FPGA and 356 DSPs on the U280 FPGA and up to 10× fewer DSPs compared to alternative designs. This efficiency is enabled by the XNOR-Shift Systolic Array (XSSA), which eliminates traditional DSP constraints.

Latency and Energy Efficiency: Fig. 7 illustrates the normalized latency and energy of *XShift* compared to DFX, FlightLLM, and GPU-based implementations. *XShift* achieves a 4.17× improvement

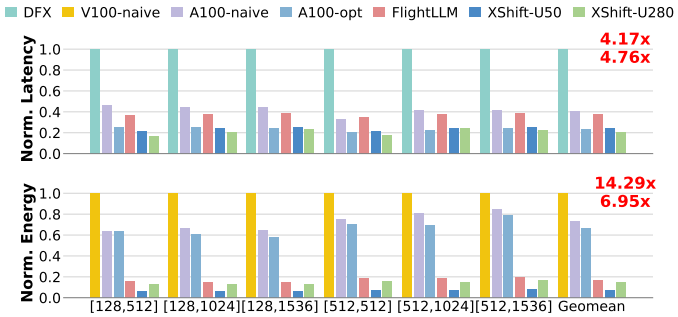


Fig. 7. Normlized latency and energy of *XShift* compared with other approaches. The horizontal axis represents [Pre-fill size, Decode size].

in latency and up to 14.29× better energy efficiency compared to the V100 GPU. Compared to the U280 FPGA implementation, the U50 FPGA achieves nearly 2× higher energy efficiency while maintaining comparable latency, emphasizing its suitability for edge deployments.

Edge Suitability: The reduced resource consumption and power requirements of *XShift* enable its deployment on cost-effective platforms like the Alveo U50 FPGA with a much better energy efficiency, making it highly practical for edge environments. Its optimized hardware-software co-design ensures that it not only meets but exceeds the performance of more resource-intensive solutions, such as FlightLLM, across both speed and energy metrics.

XShift establishes itself as a leading solution for quantizing and sparsifying LLMs, achieving state-of-the-art accuracy and efficiency. Its ability to combine low PPL, minimal OPs, and superior hardware utilization makes it a robust choice for deploying next-generation LLMs in resource-constrained environments. These results validate *XShift* as a comprehensive framework capable of delivering optimal performance on both software and hardware fronts.

VII. CONCLUSION

We introduce *XShift*, an algorithm-hardware co-design framework for efficient binarized LLM inference on FPGAs. By leveraging the XNOR-Shift Encoding (XSE) format and Hardware Adaptive Outlier and Sparsity (HAOS) techniques, *XShift* achieves significant compression while maintaining high model accuracy. Our framework integrates a custom-designed XNOR-Shift Systolic Array (XSSA) and a Base-2 SoftMax Converter (BSMC) to address the unique computational and memory demands of LLM inference, particularly in edge environments. Experimental results demonstrate *XShift*'s superiority in accuracy, computational efficiency, and energy savings, outperforming state-of-the-art methods on benchmarks like WikiText2 and C4. Its deployment on the Xilinx Alveo U280 and U50 FPGA highlights its hardware efficiency, making it ideal for edge environments.

By addressing the challenges of LLM binarization and tailoring hardware designs to the specific needs of Pre-fill and Decoding stages, *XShift* establishes itself as a robust solution for deploying LLMs in resource-constrained environments.

REFERENCES

[1] A. Radford, "Improving language understanding by generative pre-training," 2018.

[2] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[3] W. Huang, Y. Liu, H. Qin, Y. Li, S. Zhang, X. Liu, M. Magno, and X. Qi, "BiLLM: Pushing the limit of post-training quantization for LLMs," *arXiv preprint arXiv:2402.04291*, 2024.

[4] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, "BitNet: Scaling 1-bit transformers for large language models," *arXiv preprint arXiv:2310.11453*, 2023.

[5] C. Zeng, S. Liu, Y. Xie, H. Liu, X. Wang, M. Wei, S. Yang, F. Chen, and X. Mei, "ABQ-LLM: Arbitrary-bit quantized inference acceleration for large language models," *arXiv preprint arXiv:2408.08554*, 2024.

[6] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, "OmniQuant: Omnidirectionally calibrated quantization for large language models," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[7] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning (ICML)*. PMLR, 2023, pp. 38 087–38 099.

[8] T. Dettmers, R. A. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, and et al, "SpQR: A sparse-quantized representation for near-lossless LLM weight compression," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[9] X. Hu, Y. Cheng, D. Yang, Z. Yuan, J. Yu, C. Xu, and S. Zhou, "I-LLM: Efficient integer-only inference for fully-quantized low-bit large language models," *arXiv preprint arXiv:2405.17849*, 2024.

[10] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration," *Proceedings of Machine Learning and Systems (MLSys)*, vol. 6, pp. 87–100, 2024.

[11] Z. Guan, H. Huang, Y. Su, H. Huang, N. Wong, and H. Yu, "APTQ: Attention-aware post-training mixed-precision quantization for large language models," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[12] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[13] X. Ma, G. Fang, and X. Wang, "LLM-Pruner: On the structural pruning of large language models," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 21 702–21 720, 2023.

[14] E. Frantar and D. Alistarh, "SparseGPT: Massive language models can be accurately pruned in one-shot," in *International Conference on Machine Learning (ICML)*. PMLR, 2023, pp. 10 323–10 337.

[15] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang *et al.*, "FlightLLM: Efficient large language model inference with a complete mapping flow on fpgas," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2024, pp. 223–234.

[16] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "DFX: A low-latency Multi-FPGA appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022.

[17] X. Lin, H. Tian, W. Xue, L. Ma, J. Cao, M. Zhang, J. Yu, and K. Wang, "FLAME: Fully leveraging moe sparsity for transformer on FPGA," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[18] Y. Qin, Y. Wang, D. Deng, Z. Zhao, X. Yang, L. Liu, S. Wei, Y. Hu, and S. Yin, "Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023.

[19] J. Lee, W. Lee, and J. Sim, "Tender: Accelerating large language models via tensor decomposition and runtime requantization," *arXiv preprint arXiv:2406.12930*, 2024.

[20] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, and et al, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023, pp. 1–15.

[21] J. Koo, D. Park, S. Jung, and J. Kung, "OPAL: Outlier-preserved microscaling quantization accelerator for generative large language models," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[22] C. Xue, C. Zhang, X. Jiang, Z. Gao, Y. Lin, and G. Sun, "Oltron: Algorithm-hardware co-design for outlier-aware quantization of LLMs with inter-/intra-layer adaptation," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[23] Z. Liu, B. Oguz, A. Pappu, L. Xiao, S. Yih, M. Li, R. Krishnamoorthi, and Y. Mehdad, "BiT: Robustly binarized multi-distilled transformer," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 14 303–14 316, 2022.

[24] P.-H. C. Le and X. Li, "BinaryViT: Pushing binary vision transformers towards convolutional models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[25] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for Transformer-Based generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[26] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative LLM inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.

[27] M. Brysbaert, "How many words do we read per minute? a review and meta-analysis of reading rate," *Journal of memory and language*, vol. 109, p. 104047, 2019.

[28] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 688–698.

[29] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "GOBO: Quantizing attention-based NLP models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.

[30] S.-y. Liu, Z. Liu, X. Huang, P. Dong, and K.-T. Cheng, "LLM-FP4: 4-bit floating-point quantized transformers," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023, pp. 592–605.

[31] Y. An, X. Zhao, T. Yu, M. Tang, and J. Wang, "Fluctuation-based adaptive structured pruning for large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, no. 10, 2024, pp. 10 865–10 873.

[32] J. Guo, J. Wu, Z. Wang, J. Liu, G. Yang, Y. Ding, R. Gong, H. Qin, and X. Liu, "Compressing large language models by joint sparsification and quantization," in *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[33] Y. Zhang, J. Pan, X. Liu, H. Chen, D. Chen, and Z. Zhang, "FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021, pp. 171–182.

[34] S. Zhou, S. Meng, H. Tian, J. Yu, and K. Wang, "Edge-BiT: Software-hardware co-design for optimizing binarized transformer networks inference on edge FPGA," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2025.

[35] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems (NeurIPS)*, vol. 2, 1989.

[36] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[37] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems (NeurIPS)*, vol. 33, pp. 17 283–17 297, 2020.

[38] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "NN-LUT: Neural approximation of non-linear operations for efficient transformer inference," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 577–582.

[39] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016.

[40] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv e-prints*, 2019.

[41] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, and et al, "A framework for few-shot language model evaluation," 07 2024. [Online]. Available: https://zenodo.org/records/12608602

STREAMINGGS: Voxel-Based Streaming 3D Gaussian Splatting with Memory Optimization and Architectural Support

Chenqi Zhang[1*], Yu Feng[1*], Jieru Zhao[1†], Guangda Liu[1], Wenchao Ding[2], Chentao Wu[1], Minyi Guo[1]
[1] School of Computer Science, Shanghai Jiao Tong University, [2] Academy for Engineering & Technology, Fudan University
{zhangchenqi123, y-feng, zhao-jieru}@sjtu.edu.cn [*]Equal contribution. [†]Corresponding author.

***Abstract*—3D Gaussian Splatting (3DGS) has gained popularity for its efficiency and sparse Gaussian-based representation. However, 3DGS struggles to meet the real-time requirement of 90 frames per second (FPS) on resource-constrained mobile devices, achieving only 2 to 9 FPS. Existing accelerators focus on compute efficiency but overlook memory efficiency, leading to redundant DRAM traffic. We introduce STREAMINGGS, a fully streaming 3DGS algorithm-architecture co-design that achieves fine-grained pipelining and reduces DRAM traffic by transforming from a tile-centric rendering to a memory-centric rendering. Results show that our design achieves up to 45.7× speedup and 62.9× energy savings over mobile Ampere GPUs.**

***Index Terms*—3D gaussian splatting, rendering, accelerator**

I. INTRODUCTION

In virtual and augmented reality (VR/AR), photo-realistic and real-time rendering is essential for delivering immersive user experiences. Among various rendering techniques [1]–[6], 3D Gaussian Splatting (3DGS) [7]–[12] has quickly gained its popularity for its efficient rendering process and sparse Gaussian representation (also called Gaussian ellipsoids) of scene geometries.

Nevertheless, 3DGS still falls short of the real-time requirement in the VR/AR domain, namely 90 frames per second (FPS) [13]–[15], on mobile devices like VR headsets. Our experiments show that 3DGS merely achieves 2 to 9 FPS on the Nvidia Orin NX [16] [1]. Although several studies have proposed dedicated accelerators for 3DGS [18], their designs primarily optimize compute efficiency and often overlook off-chip memory traffic during rendering, making them hard to satisfy the real-time requirement in real-world scenarios.

Algorithmically, existing methods [7], [18] focus on a tile-centric rendering paradigm which requires excessive intermediate data communication between stages, leading to frequent off-chip traffic in Fig. 1a. Our characterization shows that, under a 90 FPS real-time requirement, the DRAM traffic of the current rendering paradigm would exceed the bandwidth limits of today's mobile devices on real-world scenes (Sec. II-B).

[1]The performance of the Ampere GPU (3.7 TFLOPS, released in February 2023) is comparable to that of the GPU (3.5 TFLOPS) in the latest Snapdragon XR2 [17], released in September 2023 for VR.

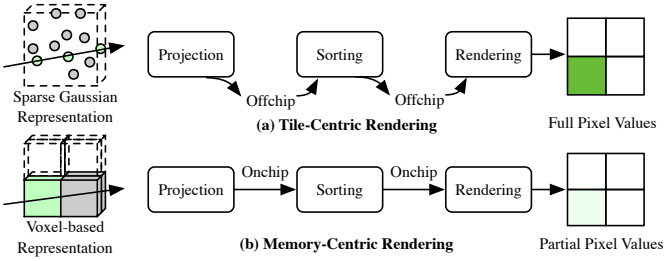


Fig. 1. Tile-centric rendering (a) vs. memory-centric rendering (b). The computation pattern is highlighted in green. Tile-centric rendering computes the complete pixel values tile-by-tile. Memory-centric rendering computes the partial pixel values voxel-by-voxel and regularizes the memory accesses.

The high data communication in the conventional tile-centric rendering comes from sequential processing ray-Gaussian intersections for each individual pixel, leading to redundant off-chip traffic and memory irregularity [18]–[21]. To address the bandwidth limitation, we argue to shift the rendering paradigm from a *tile-centric* to a *memory-centric* approach. Here, we introduce a fully streaming algorithm for 3DGS to address memory access irregularity (Sec. III). Our key idea is to regularize sparse Gaussian representations by partitioning the scene into voxels and performing on a voxel basis as shown in Fig. 1b. With all the points within a voxel stored contiguously in DRAM, we allow all points within a voxel to be streamed onto the on-chip buffer and only write the final pixel values off-chip, completely eliminating the intermediate off-chip traffic between stages.

While our memory-centric approach eliminates the intermediate off-chip traffic, our approach potentially introduces unnecessary off-chip traffic at the voxel streaming, as all points within a voxel are loaded on-chip as a whole. To reduce the unnecessary off-chip traffic, we propose hierarchical filtering to progressively exclude irrelevant Gaussians (Sec. III-B).

Specifically, we split each Gaussian feature into two halves and intentionally keep the first half lightweight. We then perform coarse yet lightweight computation to filter out irrelevant points based on the first half (i.e., the position and radius). Next, we perform precise filtering to further remove the irrelevant Gaussians by loading the second half. To further

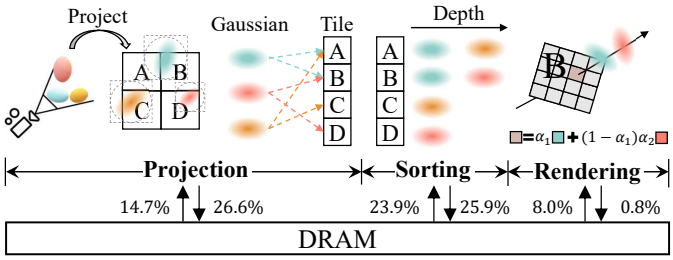


Fig. 2. The overall rendering pipeline of 3DGS. The percentage numbers represent the DRAM traffic proportion across stages.

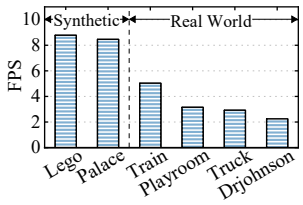


Fig. 3. FPS of different scenes on a mobile SoC, Orin NX. 3DGS cannot achieve real-time on both synthetic and real-world scenes.

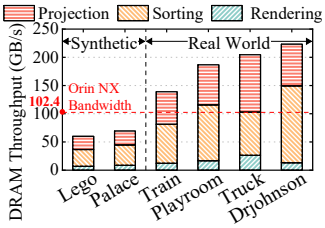


Fig. 4. The DRAM bandwidth requirement for 90 FPS under different scenes. The red dashed line highlights the bandwidth limit of Orin NX.

reduce the overhead of feature loading, we compress the second half using vector quantization. By storing a compact codebook on-chip, we just need to fetch their lightweight indices from DRAM. In this way, we drastically reduce the DRAM traffic at the voxel streaming.

To unleash the full potential of our fully streaming algorithm, we co-design an efficient accelerator (Sec. IV), including a voxel sorting unit to accelerate the determination of voxel rendering order and a dedicated hierarchical filtering unit to support Gaussian filtering at the voxel streaming. Coupled with our algorithmic optimizations, our design achieves a 45.7× speedup and 62.9× energy savings compared to a Nvidia mobile Ampere GPU [16]. Even compared against the state-of-the-art accelerator, GSCore [18], we can achieve a 2.1× speedup and 2.3× energy savings.

In summary, our contributions are as follows:

- We introduce a *memory-centric* rendering paradigm that can achieve fully streaming 3DGS rendering, completely avoiding intermediate off-chip traffic.
- We propose a hierarchical filtering scheme to further reduce the data traffic of the voxel streaming.
- With our architectural supports, our optimizations achieve an order of magnitude speedup and energy savings compared to a mobile Ampere GPU. We also achieve a 2.1× speedup and 2.3× energy savings against GSCore.

II. Background and Characterization

A. Background

Overall, 3DGS uses Gaussian ellipsoids to render a scene in a tile-by-tile fashion. Fig. 2 shows that this process can be classified into three stages:

- **Projection.** This stage projects each Gaussian ellipsoid onto the 2D screen and determines its intersected tiles as shown on the left side of Fig. 2. Each Gaussian might intersect with multiple tiles, as the colored dashed arrow lines show. Meanwhile, this stage calculates Gaussian properties, such as color and depth, for the later stages.
- **Sorting.** Each tile then sorts its intersected Gaussians based on Gaussian depths to ensure the correct rendering order from the closest to the farthest.
- **Rendering.** Every pixel within a tile traverses the same sorted Gaussian list and accumulates Gaussian colors to itself via alpha blending. For instance, we show tile B needs to blend two Gaussians, a blue one and a red one.

B. Characterization

Performance. We first show that 3DGS algorithms cannot achieve real-time performance on mobile devices. Fig. 3 characterizes the FPS of 3DGS [7] across various scenes on NVIDIA's Jetson Orin NX [16]. As the rendering scenes become more complex, shifting from synthetic to real-world scenes, performance drops from 8.5 to 4.9 FPS.

DRAM Traffic. We then show that DRAM traffic has a strong correlation with performance. Fig. 4 illustrates the DRAM throughput required to achieve the real-time requirement (90 FPS) for AR/VR applications [13], [14] across different rendering scenes. For real-world scenes, the DRAM bandwidth demand exceeds the bandwidth limit of a recently released mobile device, Nvidia Orin NX [16], as highlighted in the red dashed line. This means that the data communication requirement alone has made it impossible to achieve real-time.

We further dissect the DRAM traffic contributions across different scenes. As shown in Fig. 4, *projection* and *sorting* dominate the overall off-chip data communication, together accounting for 90% of the total DRAM traffic. The reasons that *projection* and *sorting* become the major contributors are:

- In projection, each Gaussian requires loading 59 parameters into the on-chip buffer and writing the processed features and intersection metadata back to DRAM, accounting for 41% of DRAM traffic.
- In sorting, repetitive read and write operations for progressive Gaussian sorting lead to repetitive DRAM access, taking up over 49% of DRAM traffic.

Overall, the intermediate data communication accounts for 85% of the total off-chip traffic in Fig. 2. This result motivates us to propose a fully-streaming algorithm that minimizes the off-chip traffic. Sec. III presents our algorithm design and Sec. IV explains the architectural support for our algorithm.

III. Algorithm Design

Sec. III-A first gives an algorithm overview and challenges that we need to address. Then, Sec. III-B and Sec. III-C explain the key components in our algorithm and the data layout that supports our algorithm, respectively.

A. Overview

The main downside of the original 3DGS pipeline is that it renders a frame tile-by-tile and each tile executes three stages sequentially. The reason is that the intermediate data of each

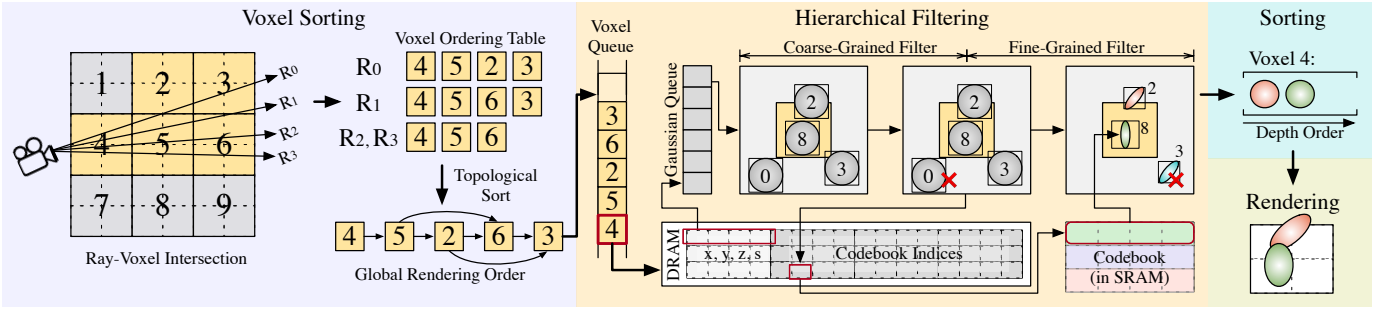


Fig. 5. The overview of our fully-streaming algorithm. Our algorithm initially partitions the entire 3D scene into small voxels. For each image tile, we first establish a rendering order of the intersected voxels via topological sort, then, process these voxels sequentially. During each voxel processing, we apply hierarchical filtering to further reduce the Gaussian loading within a voxel before the remaining sorting and rendering stages. The final pixel values are obtained by accumulating all the partial results across voxels.

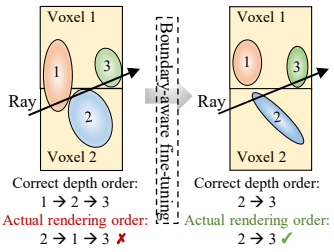


Fig. 6. An example of incorrect rendering order among Gaussians and the effect of boundary-aware fine-tuning.

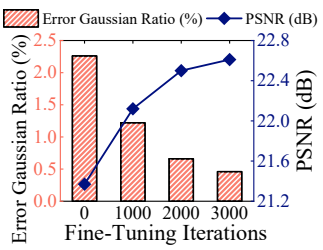


Fig. 7. Changes in the ratio of Gaussians with incorrect depth order and PSNR during the boundary-aware fine-tuning process.

stage is too large to be accommodated on-chip, inevitably, introducing frequent off-chip traffic during the execution.

Idea. To eliminate frequent off-chip communications at intermediate stages, our idea is to partition an entire 3D scene into small voxels and incrementally render the entire scene voxel-by-voxel. By processing one voxel at a time, our algorithm guarantees the intermediate data between stages can be completely held on-chip, thus, no off-chip traffic occurs.

Fig. 5 gives an overview of our algorithm. Our algorithm renders a group of pixels together, e.g., R_0 to R_3 are in the same group. For each group, we first identify which voxels intersect with these pixel rays and store the results in the voxel rendering table. Once each pixel's intersected voxels are determined, we establish their global rendering order. Then, each voxel goes through hierarchical filtering, sorting, and rendering in order, and obtain the partial pixel values of this tile. The final pixel values are obtained by accumulating all the partial results across voxels. By rendering one voxel at a time, each stage requires minimal data communication, thus, no off-chip traffic is required throughout the execution.

Challenges. However, there are three challenges when applying our algorithm:

- *Inter-Voxel Order.* Pixels within a single group often intersect with multiple voxels, as shown in Fig. 5. The rendering order for each pixel should be from the closest voxel to the farthest. The first challenge is to ensure the correct global order for all intersected voxels.
- *Cross-Voxel Gaussian Order.* Recall, each pixel is required to render Gaussians from the closest to the farthest in 3DGS, but the correct voxel-level ordering does not always guarantee the rendering order correctness at Gaussian-level granularity. For instance, on the left side of Fig. 6, Gaussian 2 in voxel 2 renders before Gaussian 1 in voxel 1, whereas the correct order should be reversed.
- *Redundant Gaussians in Voxels.* Lastly, each voxel includes Gaussians within close proximity; however, loading an entire voxel might unintentionally bring unrelated Gaussians on-chip. Leaving these redundant Gaussian without care would introduce excessive DRAM traffic.

B. Fully Streaming Algorithm

In this section, we explain how we address these challenges.

Inter-Voxel Order. To establish the correct voxel order across pixels in each tile, we first collect the rendering order of individual pixels in a voxel ordering table in Fig. 5. Based on individual rendering orders, we construct a direct acyclic graph (DAG), where each voxel is represented as a node, and edges represent the rendering dependencies. We then apply a topological sort [22] on the DAG to guarantee that no cycles in this graph and all the render dependencies are satisfied. The sorted order is then used in the remaining process.

Cross-Voxel Gaussian Order. To ensure correct rendering order among Gaussians, we observe that the incorrect order occurs only when a Gaussian spans across multiple voxels. In Fig. 6, Gaussians should be rendered from Gaussian 1 (the closest) to Gaussian 3 (the farthest). However, with voxel-by-voxel rendering, Gaussian 2 is incorrectly rendered before Gaussian 1. Fig. 7 shows that incorrect ordering can lead to a degradation in rendering quality.

To alleviate this issue, we introduce a *boundary-aware fine-tuning* to adjust cross-voxel Gaussians. Specifically, we keep each Gaussian position fixed to retain the scene geometry while fine-tuning its remaining trainable parameters, such as scale, orientation, etc. The fine-tuning reduces the size and modifies the orientation of Gaussians, minimizing their likelihood of spanning across other voxels.

In our fine-tuning, we propose a new loss function to punish the cross-boundary Gaussians,

$$\mathcal{L} = \mathcal{L}_{origin} + \beta \mathcal{L}_{CBP}, \tag{1}$$

where $\mathcal{L}_{origin}$ is the original loss in 3DGS and $\mathcal{L}_{CBP}$ is the cross-boundary penalty loss which penalizes Gaussians that span across voxel boundaries. β is a hyper-parameter to balance the influence of $\mathcal{L}_{CBP}$.

$\mathcal{L}_{CBP}$ minimizes the scale of the Gaussians that cross voxel boundaries and is expressed as,

$$\mathcal{L}_{CBP} = \frac{1}{N}\sum_{i=0}^{N-1} S_i T_i, \tag{2}$$

$$\text{where } T_i = \begin{cases} 1, & \text{if depth}_i < \max_{j\in[0,i)}\{\text{depth}_j\}, \\ 0, & \text{otherwise}, \end{cases}$$

where N is the total number of Gaussians, S_i is the maximum scale of Gaussian i. T_i is an indicator that checks if current Gaussian i has the largest depth compared to all previously rendered Gaussians. If this condition is not satisfied, we penalize it by setting T_i to be 1.

Fig. 7 illustrates the effectness of our boundary-aware fine-tuning. After 3,000 iterations, the percentage of cross-boundary Gaussians decreases from 2.3% to 0.4%, while the rendering quality regains from 21.37 dB to 22.61 dB.

Redundant Gaussians in Voxels. Recall that during projection, each Gaussian requires 59 parameters. To avoid irrelevant Gaussians, we propose a two-phase *hierarchical filtering* to progressively fetch Gaussian parameters in Fig. 5.

The first phase serves as a lightweight but coarse-grained filter that approximates whether Gaussians are intersected with the image tile. In this phase, each Gaussian only requests 4 parameters (3D coordinates and maximum scale) from off-chip memory to compute the projected center and the maximum projected radius with respect to the image tile. If a Gaussian does not pass the coarse-grained filter, we skip its remaining computations and avoid loading its remaining 55 parameters. For example, Gaussian 0 in Fig. 5 does not intersect with the image tile in yellow, we can safely remove it from the subsequent computation.

In the second fine-grained filtering, we identify the actual Gaussians that are intersected with the image tile by computing precise projection. For example, after getting shapes and orientations, we can know that Gaussian 3 in Fig. 5 does not intersect with the image tile actually and hence can skip its subsequent sorting and rendering. To further reduce the off-chip traffic, we propose a novel data compression scheme to encode the remaining parameters into a compact codebook. This way, we only need to load codebook indices from DRAM instead of the original parameters. We explain our data compression scheme in Sec. III-C. Combining coarse- and fine-grained filtering, our hierarchical filtering reduces the number of Gaussians processed in each voxel by 76.3%.

C. Customized Data Layout

To reduce data communication in hierarchical filtering, we propose a customized data layout in Fig. 8. Overall, we divide all Gaussian features into two halves, which are stored separately in DRAM and used at different filtering phases. The first half, containing 4 parameters (the 3D coordinates and maximum scale), is used for the coarse-grained filter. The second half, used in the fine-grained filter, includes the remaining parameters, such as rotation matrices.

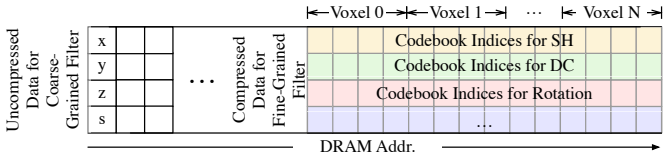


Fig. 8. The DRAM Data layout. We separate the Gaussian features into two halves for two filtering phases and compress the data for the second phase using vector quantization to reduce the off-chip communication.

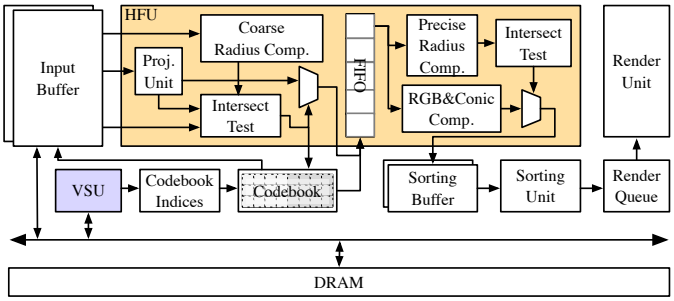


Fig. 9. Overview of StreamingGS accelerator, consisting of four components: a voxel sorting unit (VSU), a hierarchical filtering unit (HFU), a sorting unit, and a rendering unit. Our contributions are highlighted in colors: HFU is in yellow and VSU is in purple. The detailed design of VSU is in Fig. 10.

To alleviate DRAM traffic, we apply the vector quantization to compress the second half, so that we store only codebook indices off-chip while keeping the codebook itself on-chip for decoding. In addition, we encode different parameters into separate codebooks to preserve quantization precision and adopt the quantization-aware fine-tuning [9] to update the Gaussian parameters, allowing the indices to capture feature variations without loss of detail. Our result shows that, during the voxel streaming, only the codebook indices are fetched from DRAM and reduce 92.3% DRAM traffic. Note that, we only compress the second half, because we empirically find that compressing the first half leads to accuracy loss.

IV. Architectural Support

A. Overview

To support our fully-streaming algorithm, we propose our co-designed accelerator in Fig. 9. Our architecture consists of four components: a voxel sorting unit (VSU), a hierarchical filtering unit (HFU), a sorting unit, and a rendering unit.

Given a set of ray directions, VSU first identifies and sorts the intersected voxels. Then, for each intersected voxel, HFU applies a hierarchical filter to identify intersected Gaussians. Both designs will be explained in Sec. IV-B and Sec. IV-C, respectively. Once HFU obtains the filtered Gaussians, sorting and rendering units execute the sorting and rendering stage from their individual buffer, respectively. Both sorting and rendering designs are adopted from GSCore [18]. However, we simplify the sorting unit by just adopting the bitonic sorting unit from GSCore, as our voxel-based rendering only requires establishing the rendering order for Gaussians within a voxel.

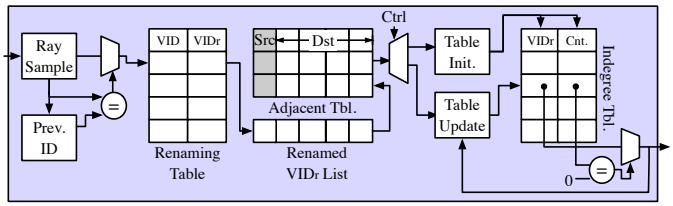


Fig. 10. The design of voxel sorting unit (VSU). It first identifies the intersected voxels for every pixel within a tile and then establishes the global rendering order of these voxels for sorting and rendering.

B. Voxel Sorting Unit (VSU)

The architecture of VSU is illustrated in Fig. 10. For pixels within a group, VSU samples along each pixel ray to identify the intersected voxels and performs a topological sort to establish the global rendering order of these voxels. Since we partition the entire scene into voxels offline, the voxel ID (VID) can be directly computed by the coordinate of a ray sample. To reduce the VID range, we further filter out those empty voxels with no Gaussians by augmenting a renaming table in VSU to reassign each VID with a new VID_r.

Each time we obtain a renamed VID_r list for one pixel, we use this list to construct an adjacent table. This adjacent table functions like a cache to help the dependency lookup among voxels. Each table entry has a tag and a value, representing a source voxel and all its corresponding destination voxels, respectively. The adjacent table is constructed once VSU finishes iterating all pixel rays in a group.

VSU then uses the adjacent table to initialize an in-degree table. Each entry in the in-degree table is directly indexed by a VID_r value and stores its in-degree count, which records the number of source voxels this VID_r has. If an in-degree of a VID_r becomes zero, this VID_r has no source dependency and can be written to the later stage. Each time VSU outputs a VID_r, VSU also updates the in-degree table accordingly. The topological sort finishes when the in-degree table is empty.

C. Hierarchical Filtering Unit (HFU)

Once VSU generates the rendering order of intersected voxels, HFU starts to load one voxel at a time and process the Gaussians within this voxel. Overall, HFU, highlighted in yellow in Fig. 9, consists of a frontend and a backend, responsible for coarse-grained and fine-grained filtering, respectively.

Coarse-Grained Filter. HFU only reads each Gaussian coordinate and maximum scale in the coarse-grained filter, computing its projected position and coarse radius. These results are sent to the intersection test unit to determine if the Gaussian intersects with the current image tile. If the condition is met, HFU signals the codebook to decode the remaining parameters from the codebook buffer, as shown in Fig. 9. The decoded parameters, along with the uncompressed parameters, are then fed into a FIFO for fine-grained filtering.

Fine-Grained Filter. This step computes the exact radius computation and performs the intersection test. Meanwhile, the fine-grained filter unit also computes the RGB values and conic matrix from decoded parameters, such as scales and rotation. Based on the intersection result, HFU conditionally outputs a Gaussian RGB and conic results to the sorting buffer if this Gaussian passes the intersection test.

In our design, the coarse-grained filter largely reduces the computation (from 427 MACs to 55), compared to fine-grained filters. In addition, the coarse-grained filter avoids decoding and reading unnecessary Gaussians to FIFO in HFU.

TABLE I
CONFIGURATION AND AREA

Unit	Configuration	Area [mm^2]
Voxel Sorting Unit	1 Unit	0.06
Hierarchical Filtering Unit	4 Units	0.79
Sorting Unit	2 Units	0.04
Rendering Unit	64 Units	2.53
SRAM (Input Buffer, Codebook, others)	355KB	1.95
Total		5.37

V. EVALUATION

A. Experimental Setup

Hardware Setup. Our accelerator consists of 1 VSU, 4 HFUs, 2 sorting units, and $4 \times 4 \times 4$ rendering units at 1GHz clock frequency. Each HFU has 4 coarse-grained filter units (CFUs) and 1 fine-grained filter unit (FFU). The sorting unit and the rendering unit are identical to the bitonic sorting unit and the volume rendering unit in GSCore, respectively [18]. The overall on-chip buffer consists of a double-buffered input buffer with a size of 16 KB, a 250 KB codebook buffer, and the remaining 89 KB to store intermediate data between stages.

Our architectural design is developed using an EDA process and synthesized with Synopsys and Cadence tools on TSMC's 32 nm FinFET technology. SRAM components are estimated by CACTI 7.0 [23] at 32 nm technology. The DRAM model in our simulations is based on Micron's 16 Gb LPDDR3 with 4 channels according to its datasheet [24], with energy consumption data sourced from Micron System Power Calculators [25].

Software Setup. We configure a codebook with 4096 entries for scale, rotation, and DC, and a codebook with 512 entries for SH coefficients in Sec. III-C. The voxel size is set to 2 for real world scenes and 0.4 for synthetic scenes. The hyper-parameter, β, in Sec. III-B is set to 0.05.

Area. STREAMINGGS design has a total of 5.37 mm^2 in Tbl. I. The major area contributors are 4 HFUs (0.79 mm^2), 64 rendering units (2.53 mm^2), and on-chip buffers (1.95 mm^2). Overall, our design has a similar area compared to GSCore (5.53 mm^2) which is scaled to 32 nm by DeepScaleTool [26].

Workloads. We evaluate both synthetic and real-world datasets: Synthetic-NSVF [27], Synthetic-NeRF [3], Tanks&Temples [28] and Deep Blending [29]. Each scene is initially trained with a default 3×10^4 iterations, followed by 3×10^3 iterations of boundary-aware fine-tuning (Sec. III-B) and 5×10^3 iterations of quantization-aware fine-tuning (Sec. III-C). We evaluated our method on three widely-adopted 3DGS algorithms, including original 3DGS [7], Mini-Splatting [8], and LightGaussian [10].

Baselines. We compare against two hardware baselines: Nvidia GPU Orin NX [16] and a dedicated accelerator, GSCore [18]. GPU performance and power consumption are

TABLE II
QUANTITATIVE COMPARISON OF RENDERING QUALITY (PSNR ↑) ACROSS DIFFERENT DATASETS AND ALGORITHMS.

Algorithm \ Dataset	Train	Truck	Playroom	Drjohnson	Lego	Palace
3DGS	22.54	26.65	30.18	29.21	36.11	38.56
Ours	22.52	26.61	30.27	29.07	36.02	38.52
Mini-Splatting	21.49	25.19	30.32	29.23	36.20	39.00
Ours	21.44	25.11	30.37	29.34	36.18	38.98
LightGaussian	22.29	26.02	28.58	25.87	35.18	37.76
Ours	22.32	25.89	28.47	25.79	35.15	37.68

obtained using built-in hardware measurements. We implement GSCore based on its published specifications and validate our results against reported data.

Variants. We evaluate three variants:

- w/o VQ+CGF performs our fully-streaming algorithm without vector quantization and coarse-grained filtering.
- w/o CGF performs our fully-streaming algorithm without coarse-grained filtering.
- STREAMINGGS is our full-fledged version including all optimization techniques.

B. Rendering Quality

Tbl. II shows a quantitative comparison of rendering quality between the original pipeline and our fully streaming pipeline across three 3DGS algorithms. Overall, our fully streaming pipeline shows no accuracy degradation in visual quality. On average, our pipeline drops the rendering quality by 0.04 dB. Our results sometimes achieve higher PSNR than the original approach. Our rendering results are also presented here: link.

C. Performance and Energy Efficiency

Fig. 11 illustrates the performance and energy savings of our design against three hardware baselines. All values are normalized against the corresponding Orin NX. Both speedup and energy numbers are averaged across four datasets.

On average, STREAMINGGS achieves 45.7× speedup against Orin NX. In comparison, GSCore can barely achieve 21.6× speedup. Our result also shows that coarse-grained filtering affects the performance of STREAMINGGS significantly. Without coarse-grained filtering, the speedup drops from 45.7× to 22.2×, while VQ has a minimal impact on the performance. This is because VQ only affects the voxel streaming which is overlapped by the subsequent stages.

On energy saving, STREAMINGGS achieves 62.9× energy reduction over Orin NX and 2.3× energy reduction over GSCore. The primary energy savings are from DRAM traffic reduction and efficient early-stage irrelevant Gaussian filtering. In addition, coarse-grained filtering and VQ contribute to 35.6× and 5.8× energy savings, respectively.

D. Sensitivity Study

Fig. 12 shows the sensitivity of rendering quality and energy savings to voxel size on one real-world scene, *train*. All variants are retrained according to our training procedure in Sec. V-A. Generally, smaller voxel sizes result in more cross-boundary Gaussians, leading to incorrect depth ordering. Our result shows that the rendering quality drops from 22.3 dB to 21.5 dB when the voxel size decreases from 2 to 0.5. However, further increasing voxel size shows minimal impacts on rendering quality. Meanwhile, larger voxels might potentially include more irrelevant Gaussians in every voxel, increasing the filtering workload and lowering energy efficiency. We empirically find a voxel size of 2 has a good balance between performance and rendering quality.

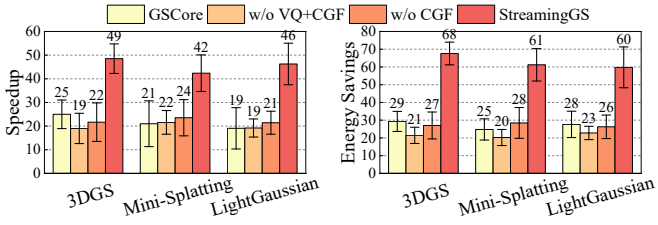


Fig. 11. End-to-end speedup and energy savings of our variants over the GPU baseline. The numbers are averaged over four datasets and normalized to the GPU baseline.

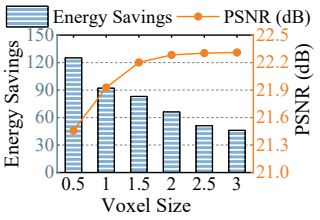


Fig. 12. The sensitivity of energy efficiency and rendering quality on the original 3DGS to voxel size. We only evaluate on *train* scene.

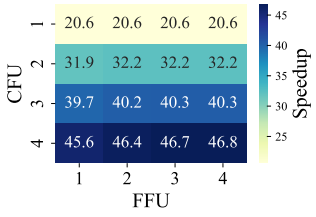


Fig. 13. The sensitivity of speedup on the original 3DGS to the numbers of CFUs and FFUs. Only *train* scene is evaluated.

Fig. 13 illustrates the sensitivity of performance to the numbers of coarse-grained filter units (CFUs) and fine-grained filter units (FFUs) in HFU (Sec. IV-C). We choose the same *train* scene and normalize all performance numbers to the GPU baseline. The results show that adding FFUs beyond the number of CFUs does not yield any speedup. In contrast, increasing the number of CFUs consistently boosts speedup, underscoring their importance in accelerating the filtering process. However, an excessive number of CFUs would lead to increased area overhead. Thus, we choose 4 CFUs with 1 FFU as our default configuration.

VI. CONCLUSION

This paper introduces STREAMINGGS that rethinks the 3DGS pipeline and proposes a new computing paradigm to process Gaussian points. Our fully streaming pipeline eliminates intermediate off-chip memory traffic and regularizes the off-chip accesses, all with a minimal but principle architecture augmentation. Overall, our algorithm-architecture co-design demonstrates 45.7× and 2.1× speedup compared to mobile GPUs and the state-of-the-art 3DGS accelerator, respectively.

VII. ACKNOWLEDGMENT

This work is sponsored by the National Natural Science Foundation of China (62472273, 62232015, 62402312).

REFERENCES

[1] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. MIT Press, 2023.

[2] Y. Deng, Y. Ni, Z. Li, S. Mu, and W. Zhang, "Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 4, pp. 1–41, 2017.

[3] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[4] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5855–5864, 2021.

[5] T. Wu, Y.-J. Yuan, L.-X. Zhang, J. Yang, Y.-P. Cao, L.-Q. Yan, and L. Gao, "Recent advances in 3d gaussian splatting," *arXiv preprint arXiv:2403.11134*, 2024.

[6] G. Chen and W. Wang, "A survey on 3d gaussian splatting," *arXiv preprint arXiv:2401.03890*, 2024.

[7] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 1–14, 2023.

[8] G. Fang and B. Wang, "Mini-splatting: Representing scenes with a constrained number of gaussians," *arXiv preprint arXiv:2403.14166*, 2024.

[9] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park, "Compact 3d gaussian representation for radiance field," *arXiv preprint arXiv:2311.13681*, 2023.

[10] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps," *arXiv preprint arXiv:2311.17245*, 2023.

[11] W. Lin, Y. Feng, and Y. Zhu, "Metasapiens: Real-time neural rendering with efficiency-aware pruning and accelerated foveated rendering," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 669–682, 2025.

[12] X. Huang, H. Zhu, Z. Liu, W. Lin, X. Liu, Z. He, J. Leng, M. Guo, and Y. Feng, "Seele: A unified acceleration framework for real-time gaussian splatting," *arXiv preprint arXiv:2503.05168*, 2025.

[13] "Meta Quest Pro specs."

[14] "Apple Vision Pro screen refresh rate is up to 100Hz."

[15] J. Wang, R. Shi, W. Zheng, W. Xie, D. Kao, and H.-N. Liang, "Effect of frame rate on user experience, performance, and simulator sickness in virtual reality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 5, pp. 2478–2488, 2023.

[16] "Nvidia jetson orin nx."

[17] "Qualcomm QCS8550/QCM8550 Processors."

[18] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim, "Gscore: Efficient radiance field rendering via architectural support for 3d gaussian splatting," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 497–511, 2024.

[19] Y. Feng, W. Lin, Z. Liu, J. Leng, M. Guo, H. Zhao, X. Hou, J. Zhao, and Y. Zhu, "Potamoi: Accelerating neural rendering via a unified streaming architecture," *ACM Transactions on Architecture and Code Optimization*, 2024.

[20] H. He, G. Li, F. Liu, L. Jiang, X. Liang, and Z. Song, "Gsarch: Breaking memory barriers in 3d guassian splatting training via architectural support," in *2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2025.

[21] Y. Feng, Z. Liu, J. Leng, M. Guo, and Y. Zhu, "Cicero: Addressing algorithmic and architectural bottlenecks in neural rendering by radiance warping and memory optimizations," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 1293–1308, IEEE, 2024.

[22] A. B. Kahn, "Topological sorting of large networks," *Communications of the ACM*, vol. 5, no. 11, pp. 558–562, 1962.

[23] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[24] "Micron 178-Ball, Single-Channel Mobile LPDDR3 SDRAM Features."

[25] "Micron System Power Calculators."

[26] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2021.

[27] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15651–15663, 2020.

[28] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, pp. 1–13, 2017.

[29] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Transactions on Graphics (ToG)*, vol. 37, no. 6, pp. 1–15, 2018.

Ares: High Performance Near-Storage Accelerator for FHE-based Private Set Intersection

Haoxuan Wang*†§, Yinghao Yang*†, Jinkai Zhang*†, Hang Lu*†§, Xiaowei Li*†§
*SKLP, Institute of Computing Technology, Chinese Academy of Sciences,
†University of Chinese Academy of Sciences,
§Zhongguancun Laboratory, Beijing, China
{wanghaoxuan23p, zhangjinkai22z, luhang, lxw}@ict.ac.cn, yinghao.y@foxmail.com

***Abstract*—Nowadays, the importance of data privacy protection has grown significantly. Privacy Set Intersection (PSI) based on Fully Homomorphic Encryption (FHE) is widely applied in various privacy protection scenarios, such as federated learning and password verification. Nevertheless, the substantial computational demands of FHE and the vast scale of databases in PSI result in inefficient processing, thereby necessitating specialized accelerator architectures to enhance usability. Current general-purpose FHE accelerators do not adequately address the unique requirements of PSI applications, leading to suboptimal data handling and underutilization of hardware, which impedes their effective deployment for PSI acceleration. This paper introduces Ares, a practical hardware-software co-designed FHE-based PSI FPGA accelerator. We propose Lazy Relinearization to optimize redundant calculations in PSI and reduce computational complexity without changing the PSI protocol. At the same time, through the analysis and decoupling of the PSI computing pattern, we design an efficient hardware acceleration architecture that fully utilizes the bandwidth and computing resources of the hardware to achieve excellent acceleration performance. We highlight the following result: (1) a 47.99× speedup relative to CPU; (2) performance improvements of 1.79× and 1.93× over the state-of-the-art FPGA FHE accelerators, Poseidon and FAB, respectively; (3) achieves 7.96× and 10.95× energy efficiency improvement compared to Poseidon and FAB, respectively.**

I. Introduction

In the era of big data, the demand for efficient and secure data processing has become paramount. The concept of privacy computation was proposed to address the issue of data privacy breaches in complex environments. PSI is a foundational cryptographic protocol in privacy computation, allowing two parties to determine the intersection of their sets without revealing any additional information. Among various implementations, the PSI implementation based on FHE is the most widely used. The HE-based PSI protocol can perform set intersections while data remains encrypted. For instance, Microsoft integrated this technology into its Edge browser to detect leakage of user passwords [1]. Similarly, Apple released a homomorphic encryption library in Swift, which it employs in the latest iPhone to support Live Caller ID Lookup functionality [2]. In real-world PSI applications, the database size often easily reaches the level of the billions [3]. Such large computational workloads and databases make it impractical for general-purpose hardware such as CPUs to perform FHE-Based PSI.

Currently, many dedicated FHE accelerators—such as ARK [4], SHARP [5], CL [6], Poseidon [7] and FAB [8] have been developed to accelerate FHE computations. These architectures can also be applied to HE-based PSI. Fig. 1 illustrates the architecture of a traditional FHE accelerator and its main components and processes involved in PSI computation. However, employing these general FHE accelerators to perform PSI queries directly would be extremely inefficient.

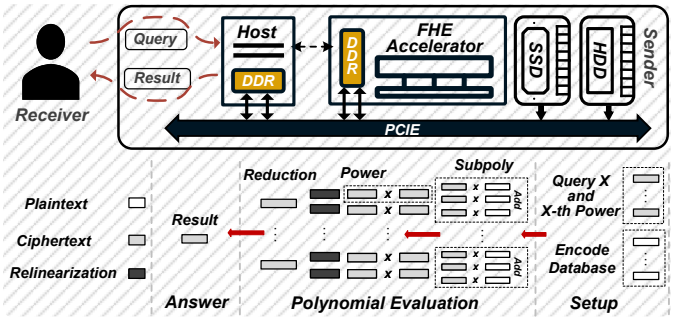


Fig. 1. The data path for executing PSI query with general FHE accelerators. These accelerators are typically connected to the host system as PCIe expansion cards, meaning that when processing large-scale datasets, data must first be read from external storage devices (e.g., SSDs or HDDs).

Firstly, due to the PSI protocol's requirement to perform computations on each entry in the database, large-scale datasets cannot be entirely stored in the onboard memory of FHE accelerators. Instead, they must be stored on the host's SSD or HDD and transferred to the accelerator via the PCIe interface during computation. Although DMA can reduce CPU involvement by facilitating data transfer between the SSD and host, a large portion of execution time is still spent on data transfer over the busy (heavy-load) system PCIe bus.

Secondly, another limitation of using general-purpose FHE accelerators is their low hardware utilization. These architectures are designed to accelerate generic FHE schemes without considering the specific requirements of PSI applications. The PSI query flow, as illustrated in the lower part of Fig. 1, shows that the database is encoded into plaintext polynomial form, where the key computation is polynomial evaluation. This process requires only a subset of FHE computational functions, leading to inefficient use of both computational and bandwidth resources in prior FHE accelerators. Therefore, FHE-based PSI requires specialized hardware acceleration architecture to optimize performance according to its requirements.

Furthermore, the computational pattern in PSI protocols exhibits some redundancy. As shown in the lower part of Fig. 1, the polynomial evaluation involves a relinearization step immediately following each ciphertext multiplication. These two steps are among the most time-consuming in FHE. However, based on the computational pattern of the PSI, relinearization does not need to be performed immediately after every ciphertext multiplication. Therefore, to improve the efficiency of FHE-based PSI, it is necessary not only to conduct targeted designs at the hardware level but also to optimize

Haoxuan Wang and **Yinghao Yang** are co-first authors. **Hang Lu** is the corresponding author.

the computation pattern at the software level.

In this paper, we propose Ares, an FHE-based PSI accelerator solution with hardware-software co-designed based on Near-Data Processing (NDP). We introduce the lazy relinearization optimization strategy to reduce the complexity of polynomial evaluation. Building on this, we thoroughly explore the characteristics of PSI computations and design a corresponding hardware accelerator architecture, which significantly improves the performance of FHE-based PSI queries, making them more suitable for practical applications.

In summary, our contributions are as follows:

- We introduce an optimization termed Lazy Relinearization, which aims to reduce computational complexity by minimizing redundant and costly relinearization operations during the PSI computation process.
- We propose Ares, a dedicated HE-based PSI accelerator architecture. This architecture is optimized for computational patterns and fully adapts to PSI's characteristics, significantly enhancing the efficiency of PSI queries.
- We implemented a prototype of Ares on the commercial NDP platform SmartSSD. The results show that Ares has a 48× acceleration compared to the CPU. Compared to the SOTA FPGA FHE accelerator, it offers a 1.93× improvement in performance and a 10.95× improvement in energy efficiency.

II. Background & Motivation

A. FHE-Based PSI

In a classic FHE-based PSI [9]–[11], the sender holds a larger set X with size $|X|$, and the receiver holds a smaller set Y with size $|Y|$. The receiver encrypts its set $Y = \{y_1, y_2, \ldots, y_{|Y|}\}$ into ciphertexts $Y' = \{c_1, c_2, \ldots, c_{|Y|}\}$. The sender computes an interpolating polynomial $F_X(c) = \prod_{m \in X}(m - c)$ for each c_i, which is the key computation in PSI and also the most complex calculation. At the end of the computation, the receiver decrypts the result: $F_X(c_1), F_X(c_2), \ldots, F_X(c_{|Y|})$, and if the result of any $F_X(c_i)$ is zero, the corresponding y_i is an element of the intersection $X \cap Y$.

The FHE-based PSI protocol is typically implemented using the BFV scheme [12]. In BFV, When calculating $F_{X_1}(c)$, $F_{X_2}(c)$, …, $F_X(c_{|Y|})$, we take $F_{X_1}(c)$ as an example. $F_{X_1}(c) = \sum_{i=0}^{D} a_i c^i$, where D stands for the degree of $F_{X_1}(c)$, a_i is the encoded BFV plaintext of the database, and c represents the encoded BFV ciphertext of the user's set Y. The $F_X(c)$ is called polynomial evaluation.

In polynomial evaluation, both plaintexts and ciphertexts are represented by polynomials. The number of polynomials in a ciphertext or plaintext is denoted as N_p. Initially, N_p is 2 for ciphertexts and 1 for plaintexts. During homomorphic multiplication(Hmul), the N_p of the result is the sum of the N_p values of the two inputs minus one. For instance, if $A = (c_0, c_1)$ and $B = (d_0, d_1)$, then $A \times B = (c_0 d_0, c_0 d_1 + c_1 d_0, c_1 d_1)$. To manage the growth of the polynomial number N_p, relinearization (relin for short) is used to reset the number back to 2.

It can be seen that FHE-based PSI essentially consists of a series of polynomial evaluation computations. Therefore, in this paper, we focus on optimizing and accelerating the polynomial evaluation computations on the sender's side.

B. Redundancy in PSI computation pattern

In the BFV scheme, to prevent the exponential growth of the number of polynomials in ciphertexts during cascaded computations, the relin operation is typically performed immediately after each Hmul. However, in FHE-based PSI, the depth of operation cascades is limited. As shown in Fig. 1, the Hmuls within the polynomial evaluation in FHE-based PSI usually occur at the same depth and do not result in an exponential increase in the number of polynomials. Therefore, performing a relin operation immediately after each ciphertext multiplication is unnecessary in PSI.

Taking the current state-of-the-art (SOTA) APSI protocol [13] as an example, it also fails to recognize this fully and continues to perform a relin after every Hmul. Since Relin is one of the most time-consuming operations in the BFV scheme, this results in significant redundant computation. In Ares, we propose the Lazy Relinearization optimization for this phenomenon, which will be detailed in Sec.III-A.

C. Mismatch with the General FHE Accelerators

There are two obvious limitations to directly using general-purpose FHE accelerators to implement PSI applications.

Low hardware utilization. Traditional accelerator architectures are built to handle large encryption parameters (e.g., $N = 2^{16}$, $\log Q$ over 1500 bits) and complex bootstrapping operations. To support these requirements, they must deploy substantial scratchpad memory (180MB∼512MB) and high-bandwidth off-chip memory (e.g., two HBM modules with 1TB bandwidth) to meet the data R/W demands of the computation units. However, PSI requires significantly smaller encryption parameters (e.g., $N = 2^{13}$, $\log Q$ less than 220 bits), so using these accelerators results in considerable waste of storage and bandwidth resources. Moreover, PSI employs only a subset of FHE operations, excluding functions such as automorphism and bootstrapping, which are commonly supported by general-purpose accelerators. This further reduces hardware efficiency.

Significant data transfer overhead. These accelerators typically face memory limitations when using traditional memory hierarchies, meaning the entire database cannot be fully loaded into on-board memory. However, SSD bandwidth is much lower than that of memory. The commonly used PCIe 3.0×4 interface in SSDs usually has a bandwidth of only about 1.2 GiB/s for this fine-grained ciphertext read operation [14]. During computation, each query requires simple homomorphic operations (i.e., ciphertext addition) with all entries in the dataset (encoded plaintext), and each piece of data is used only once per query. This results in the database being used briefly with no opportunity for reuse. It is difficult to simultaneously transmit the next set of database during computation to mask the significant PCIe communication overhead.

III. Hardware-Software Co-Design for Ares

In this section, we first explore how Lazy Relinearization optimization can effectively reduce the relin overhead in PSI computations, thereby enhancing efficiency. Next, we present design strategies for optimizing the HE computation data flow in PSI, aiming to increase the throughput of PSI computations. Subsequently, we provide a detailed description of the Ares hardware architecture, which is based on these optimizations. Finally, we analyze how the optimized PSI computation workload is efficiently executed on the Ares hardware architecture, ensuring high performance and energy efficiency even when processing large-scale databases.

A. Lazy Relinearization

As discussed in Sec.II-B, there exists a certain amount of redundant computation in the polynomial evaluation. We propose an optimization strategy named Lazy Relinearization (LazyRelin for short), which aims to eliminate these unnecessary computations.

The upper part of Fig. 2 illustrates an example procedure for polynomial evaluation in previous PSI implementations. Here, the

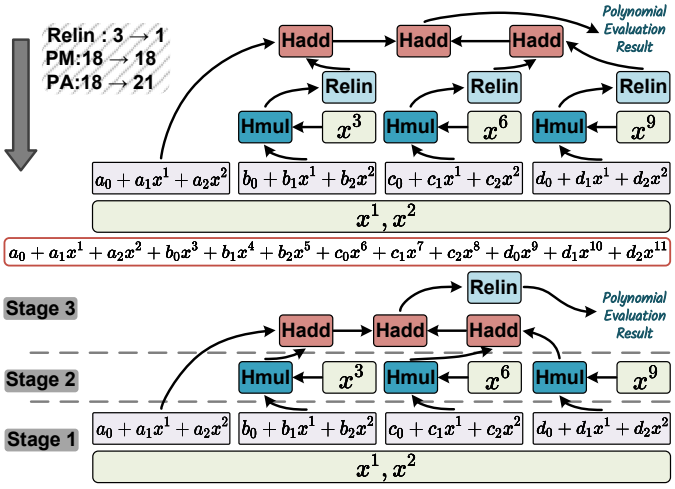


Fig. 2. Example of LazyRelin Optimization. The figure demonstrates the changes in polynomial evaluation computations before and after applying LazyRelin. The red box highlights the polynomial that requires evaluation.

red box denotes a 12th-degree polynomial to be evaluated, where PM stands for polynomial multiplication and PA for polynomial addition. The polynomial is initially decomposed into several sub-polynomials to reduce the required computational power. Notably, in these sub-polynomials, the query x and its powers are in ciphertext form, while other terms are in plaintext. The difference is in Stage ❷, after completing Hmul, we do not perform the relinearization immediately. Instead, we keep the number of polynomials at three until all Hadd aggregations are finished. Then, we perform a single relinearization on the final result.

As mentioned earlier, Hmul operations performed at the same depth do not cause a surge in polynomial numbers. Since all Hmul operations in the polynomial evaluation of PSI occur at the same depth, delaying relinearization is both feasible and merely alters the computation pattern without affecting the accuracy of the PSI protocol execution.

In this example, the polynomial evaluation process eliminates two time-consuming relinearization steps, with the trade-off being a minor increase in PAs, which has negligible overhead. The experimental results of applying this optimization strategy to practical-scale PSI will be presented in Sec.IV-C

B. Architecture

Ares is a NDP architecture that integrates storage, memory, and computing within a single Computational Storage Device (CSD), effectively alleviating bottlenecks associated with data movement over a busy system PCIe bus. The Fig.3 (a) illustrates the overall architecture of Ares. Within the CSD, the programmable circuit contains the computational architecture. And, the remaining components of the CSD consist of DRAM and SSD storage units, along with a PCIe switch. The PCIe switch facilitates communication not only between the host and the device but also supports point-to-point (P2P) communication between the computing circuits and the SSD storage. Databases stored in the SSD are first cached in DRAM before being read into two regions designated for computation, which are partitioned according to the optimized PSI computation pattern. These regions transfer data through a unidirectional Polynomial Evaluation Intermediate Result (PEIR) buffer.

As discussed in Sec.II-A, the primary computational workload in PSI is concentrated in the polynomial evaluation. The database is stored in plaintext form and processed with the query ciphertexts. As shown in the lower half of Fig.2, Stage ❶ involves the computation of sub-polynomials, which includes operations such as plaintext multiplication (Pmul) and ciphertext addition (Hadd). In contrast, Stage ❷ focuses on Hmul, a process with significantly higher computational complexity—typically two orders of magnitude more complex—than the operations in Stage ❶.

Based on these observations, the polynomial evaluation can be decomposed into two components: one is memory-intensive (Pmul and Hadd) and another is compute-intensive (Hmul and relinearization). These two components are computationally independent but exhibit memory locality in their data dependencies. The result of the former is just the input of the latter. However, traditional general-purpose FHE accelerators do not incorporate specialized data pathways tailored to the requirements of polynomial evaluation in PSI, thereby failing to fully leverage the benefits of this decoupling in terms of memory access and computation. Ares could optimize the parallelism and locality inherent between the two components.

As shown in the right half of Fig.3 (a), we divide the computation into two parts in the hardware architecture: the *Memory-Bound (MB) Region* and the *Computation-Bound (CB) Region*. This unidirectional dataflow minimizes redundant data transfer, aligning closely with the characteristics of PSI computations to reduce pressure on memory access. After an assignment of computational resources, these two parts achieved a balance in performance, and a high-performance bubble-free pipeline architecture was implemented with some buffer. The advantages of this unidirectional dataflow will be seen in Sec.III-C. Below we will introduce the Ares computing architecture in detail.

MB Region is responsible for executing Pmul and Hadd. Although these operations have low computational complexity but need to be executed an extremely large number of times, thus requiring high memory bandwidth. Data is retrieved from DRAM and processed through cascaded Pmul and Hadd units to complete sub-polynomial evaluation calculations. The results of these evaluations are temporarily stored in a PEIR buffer before being forwarded to the CB Region for subsequent processing.

CB Region is much more complex. It can be subdivided into two primary components: one module dedicated to Hmul and another unit for the accumulation of multiple sub-polynomials, known as the HAccumulate unit. The hardware design for Hmul includes modular multiplication (MM), modular addition (MA), Number Theoretic Transform (NTT), and a separate Evaluation Key (EVK) buffer. The modular multiplication operation employs Barrett reduction, while the NTT unit adopts an 8-Radix structure inspired by the Poseidon architecture. The HAccumulate unit is a higher-performance variant of the Hadd unit equipped with independent caching. Unlike the Hadd units in the MB Region, the ciphertext polynomials have not undergone relinearization, leading to a larger input dimension for the Hadd computation. The accumulated unrelin results are then transferred via a Local Buffer. Notably, since the basic operators for relin and Hmul computations are identical, coupled with the use of relinearization keys stored in the EVK buffer, the relin process can be completed, with the final output written back to DDR through the outbuffer.

PEIR Buffer can be conceptualized as a dual-port asynchronous FIFO designed to cache the ciphertext results of sub-polynomial computations and transmit them sequentially to the CB Region. Given the generally balanced performance between the two regions, the buffer's capacity does not need to be extensive. In our design, it is configured to hold three ciphertexts.

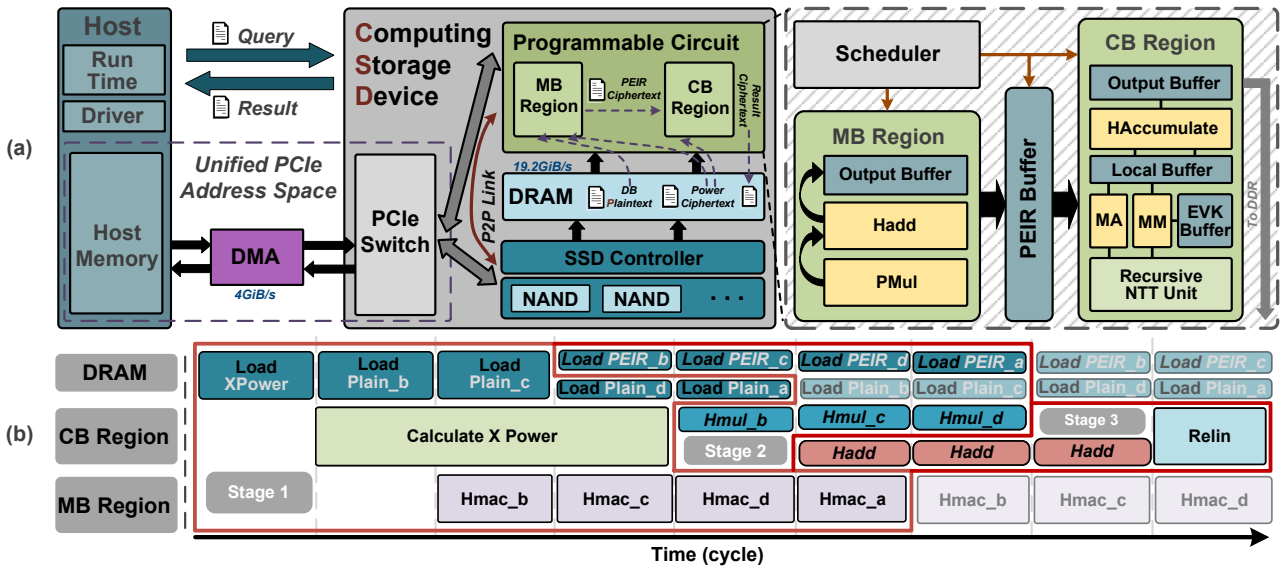


Fig. 3. The left half of part (a) shows the collaborative process between Ares and Host to complete a PSI query, while the right half is the overall computing architecture of Ares. Part (b) illustrates how PSI workload is allocated and executed on two specific regions under the hardware architecture of Ares.

Scheduler is implemented as a finite state machine (FSM) that is responsible for communicating with the Runtime on the Host side. Its primary functions include receiving the ciphertext of the query x and its powers from the host, initiating the transition of the computational unit's state. Upon query completion, the resulting ciphertext is returned to the host.

C. Workload Distribution to different Regions

In Sec.III-A, we explain how the optimized PSI computation pattern performs polynomial evaluation. This section further elaborates on how our hardware design effectively supports these optimizations in conjunction with the Ares hardware architecture. Through a co-design approach that integrates both software and hardware, we have divided the core computational process of polynomial evaluation in PSI into three stages. As shown in Fig.3 (b), we provide a typical polynomial evaluation that illustrates the workload distribution when evaluating a 12th-degree polynomial. This example serves as an extension and concretization of the content described in the lower half of Fig.2.

Stage ❶: At this stage, the plaintext database and the power of x are read from DRAM. The polynomial is divided into several sub-polynomials; for example, *Plain_b* represents b_0, b_1, b_2 in Fig.2. The CB Region calculates the required powers of x, and for the same user, the calculation of x's power needs to be performed only once throughout the process. Meanwhile, the MB Region begins to calculate the sub-polynomials; for instance, *Hmac_b* indicates calculating $b_0 + b_1x^1 + b_2x^2$, and then the result is written into the PEIR buffer.

Stage ❷: This stage focuses on performing efficient Hmul. This part involves multiplying the results of computing sub-polynomials such as *PEIR_b* by x^3, x^6, and x^9 in *Hmul_b*, *Hmul_c*, and *Hmul_d* respectively.

Stage ❸: In the final stage, *Hadd* is utilized to aggregate the results of *Hmul*. Subsequently, only one *Relin* computation needs to be performed on this result to convert the high-order polynomial back into a decryptable form, thereby yielding the complete polynomial evaluation result.

This example remains effective when extended to practical PSI polynomial evaluations. For instance, in the evaluation of a polynomial of degree 2000, during Stage ❶, the degree of the sub-polynomials that need to be computed is 40. For Stage ❷, the required powers of x are x^{40}, x^{80}, etc. Through hardware design, operations such as Hadd, Hmul, and Relin are balanced in performance to ensure that the pipeline can execute efficiently.

There is another subtle optimization worth noting. Observing the order in which plaintext database reads from DRAM, it's evident that the reading of sub-polynomial *plain_a* is not performed at the beginning as one might intuitively expect, but rather delayed until the end of the entire read process. As shown in Fig. 2, after sub-polynomial *plain_a* completes *Hmac_a*, it does not need to participate in the *Hmul* operation in Stage ❷. Therefore, by reading sub-polynomial *plain_a* last, we can directly use it in the *Hadd* computation of Stage ❸ immediately after completing *Hmac_a*, thus better utilizing the limited bandwidth resources. Although the effect of this optimization may not be significant during a single polynomial evaluation. However, when dealing with large-scale databases, where PSI computation involves thousands of polynomial evaluations, this fine-tuning can cumulatively have a positive impact on overall performance.

IV. Evaluation

A. Experimental Setup

Software Platform: We conducted the software performance testing of Ares on a server platform equipped with an Intel Xeon W-1370 CPU, which features 16 threads. The server is also outfitted with 64 GB of DDR4 RAM and 4 TiB of SSD storage.

Hardware Platform: We implement Arse's hardware architecture in SAMSUNG SmartSSD device [15]. The SmartSSD has a 4TB NVMe SSD which directly communicates with Kintex UltraScale+ KU15P FPGA through a PCIe Gen3.0 x4 bus. The attached FPGA has approximately 522K LUTs, 984BRAMs, 1968 DSPs, and DDR4 4GB DRAM.

Baseline: In our experiment, we first evaluate the performance of the basic CPU. Subsequently, we compared Ares with SOTA FPGA-

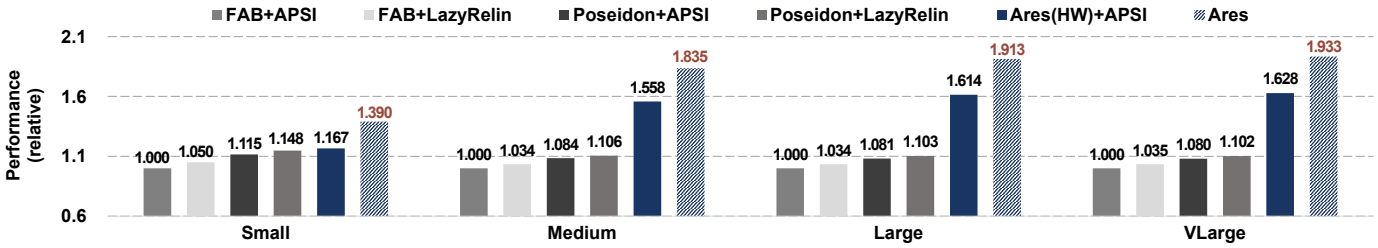


Fig. 4. The ablation experiment demonstrates the performance of LazyRelin optimization on both a general-purpose FHE accelerator and on Ares, across all four benchmarks of different scales.

TABLE I
SIZES OF BENCHMARK DATABASES

Database	Receiver.Num	Sender.Num	DB Size	Entry Length
Small	4096	2^{28}	1.68GB	16B
Medium	4096	2^{30}	6.75GB	16B
Large	4096	2^{32}	27GB	16B
VLarge	4096	2^{34}	108GB	16B

TABLE II
PERFORMANCE COMPARISON OF CPU AND FPGA-BASED FHE ACCELERATORS ACROSS FOUR BENCHMARKS

Platforms		**Small**(s)	**Medium**(s)	**Large**(s)	**VLarge**(s)
CPU		61.82	183.84	711.63	2822.79
FPGA	FAB	1.97	7.19	28.52	113.82
	Poseidon	1.77	6.63	26.38	105.38
	Ares	**1.42**	**3.92**	**14.91**	**58.88**

based FHE accelerators like FAB [8] and Poseidon [7]. None of these accelerators have implemented the PSI protocol, but each accelerator is fully capable of executing PSI. To facilitate comparison, we evaluated each accelerator architecture according to the complexity of the benchmark.

Benchmark: As shown in TableI, we selected four databases of different sizes for evaluation. The size of the receiver (user) set in all four benchmarks is 4096, so the polynomial size N for BFV is configured to 8192. The DB Size refers to the size of the database after undergoing cuckoo hashing and OFPR. For the purpose of experimental evaluation, the specific values in these four databases are randomly generated.

B. Performance

As illustrated in TableII, we initially present an overall system performance analysis for executing identical PSI queries across four databases of varying sizes, using Ares, CPU, and two SOTA FPGA-based FHE accelerators. Ares demonstrates an approximate 47× speedup over the CPU across all four datasets, primarily because HE operations constitute the primary bottleneck for PSI queries under the CPU architecture, rather than memory access becoming the limiting factor. In contrast, for smaller-scale datasets, both Ares and FPGA-based accelerators experience performance enhancements constrained by certain memory access limitations, resulting in Ares achieving only a 1.38× speedup. For larger-scale datasets, specifically Large and VLarge, Ares leverages its advantages in co-designed software and hardware to exhibit nearly double the performance improvement relative to FPGA-based accelerators.

C. Ablation

Due to the excessive resource consumption of FPGA Baseline FAB and Poseidon, they cannot be deployed on the resource-limited CSD platform. We evaluated the performance improvement of deploying LazyRelin optimization on these architectures based on their design complexity. In Figure 4, we normalized the data to the performance of FAB+APSI. The results show that, across four different benchmarks, the performance improvement of the two baselines with LazyRelin optimization was only approximately 3%, exhibiting a similar minor enhancement trend in each benchmark. This limited improvement is attributed to the general FHE accelerator architecture's inability to effectively align with the specific computational patterns of PSI.

In contrast, even using the original APSI computation patterns, the Ares architecture still performs better than the general FHE accelerator. Moreover, after applying the LazyRelin optimization, Ares's performance was further significantly improved, especially in the VLarge database tests, where the maximum performance achieved up to 1.933×. These results not only highlight the high compatibility between the LazyRelin optimization and the Ares architecture but also indirectly confirm the high alignment of the Ares architecture with the PSI computation patterns in its design.

D. Resources And Utilization

Ares exhibits significant efficiency in resource utilization. As shown in the left half of TableIII, a comparison is made between Ares and two other FPGA-based FHE accelerators regarding their primary resource consumption. The data indicate that Ares achieves a reduction of 2∼5× compared to previous FPGA accelerators, whether it pertains to the usage of logical or storage resources. The right half of TableIII evaluates the pure computational performance of the core calculation in PSI query—polynomial evaluation, where D represents the degree of the polynomial. The findings reveal that, despite a marked reduction in hardware resource usage, Ares performs slightly lower than Poseidon in polynomial evaluation but is comparable to FAB. This phenomenon can be attributed to the fact that traditional FPGA-based FHE accelerator designs have not adequately considered the characteristics of PSI computations, lacking dedicated data paths optimized for PSI tasks. Consequently, even with substantial investments in computational and storage resources, the actual utilization rate of hardware remains low. In contrast, Ares effectively maximizes the performance of PSI computations through the optimal use of limited hardware resources, demonstrating its advanced and targeted design approach.

TABLE III
RESOURCE UTILIZATION ANALYSIS: COMPARISON OF RESOURCE USAGE BETWEEN ARES AND BASELINE FPGA ACCELERATORS; POLYNOMIAL EVALUATION LATENCY

Works	Resources				Latency(ms)	
	LUT(K)	FF	BRAM	DSP	D=2^{14}	D=2^{18}
FAB	899	2,073	3,840	5,120	94.4	1511.3
Poseidon	728	977	2,048	8,640	61.3	981.8
Ares	**343**	**474**	**912**	**1,728**	80.5	1288.7

TABLE IV
EFFICIENCY ANALYSIS. ENERGY-DELAY PRODUCT (EDP) IS USED AS THE METRIC, WHERE LOWER IS BETTER. MAX GAIN INDICATES THE MAXIMUM GAIN ACROSS FOUR BENCHMARKS.

Works	Small	Medium	Large	Vlarge	Max Gain
FAB	464.15	6,182	97,281	1,549,417	**10.95**
Poseidon	317.67	4,457	70,564	1,126,041	**7.96**
Ares	**82.26**	**626**	**9,070**	**141,447**	**1.0**

E. Energy Efficiency

To evaluate Ares' performance in terms of energy efficiency, we adopted the Energy-Delay Product (EDP) as the metric. The power consumption data for FAB and Poseidon were obtained through the Xilinx Power Estimator (XPE) [16], while Ares' power consumption data were derived from Power Reports generated after synthesis using the Vitis Development Tool (VDT). It is worth noting that the EDP data for the CPU are not listed in the table because its EDP value is significantly higher—approximately 5500× greater than the accelerators, hence not directly represented in the table. As shown in TableIV, Ares demonstrates superior energy efficiency across four benchmarks, achieving a 10.95× increase in energy efficiency compared to FAB and a maximum 7.96× improvement over Poseidon. On one hand, these notable gains in energy efficiency can be attributed to Ares' efficient utilization of hardware resources, which allows it to maintain high-performance PSI computation capabilities even at lower power levels. On the other hand, the gains are also due to its architecture specifically designed for PSI computations and optimized data transfer paths, further enhancing Ares' competitive edge in energy efficiency.

V. CONCLUSION

In this paper, we present Ares, a hardware-software co-designed accelerator for FHE-based PSI. We optimize the computation pattern using the LazyRelin strategy, which significantly reduces the number of relinearization operations, one of the most time-consuming operations in FHE computations. In terms of hardware architecture design, Ares leverages NDP to minimize data transfer latency and designs matching hardware computational circuits tailored to the PSI computation pattern, thereby maximizing the accelerator's performance.

The proposed Ares accelerator not only enhances the performance and energy efficiency of FHE-based PSI but also paves the way for more practical and scalable applications in real-world scenarios. Overall, our work demonstrates the feasibility and advantages of specialized hardware acceleration for FHE-based PSI, making it a promising solution for large-scale, privacy-preserving data processing. We hope that the progress made in this research will contribute to the broader field of privacy-preserving computation.

VI. ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 62172387; in part by the CCF-Phytium Fund 2023.

REFERENCES

[1] "Password monitor: Safeguarding passwords in microsoft edge," https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/, Jan. 2021, microsoft Research.

[2] "Understanding how live caller id lookup preserves privacy," https://docs.developer.apple.com/documentation/sms_and_call_reporting/understanding_how_live_caller_id_lookup_preserves_privacy, apple.

[3] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1447–1464.

[4] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.

[5] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[6] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 173–187.

[7] Y. Yang, H. Zhang, S. Fan, H. Lu, M. Zhang, and X. Li, "Poseidon: Practical homomorphic encryption accelerator," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 870–881.

[8] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 882–895.

[9] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1243–1255.

[10] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled psi from fully homomorphic encryption with malicious security," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1223–1237.

[11] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 515–530.

[12] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[13] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg, "Labeled psi from homomorphic encryption with reduced computation and communication," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1135–1150.

[14] L. Daoud, G. Sun, and H. Huen, "Peer-to-peer data transfer evaluation in smartssd-based multi-devices system," in *Proceedings of 36th International Conference on Computer Applications in Industry and Engineering*, vol. 97, 2024, pp. 72–80.

[15] P. Mehra, "Samsung smartssd: Accelerating data-rich applications," *Flash Memory Summit*, 2019.

[16] "AMD Power Estimator." [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/power-efficiency/power-estimator.html

DenSparSA: A Balanced Systolic Array Approach for Dense and Sparse Matrix Multiplication

Ziheng Wang[1], Ruiqi Sun[1], Xin He[2], Tianrui Ma[3], and An Zou[1]
[1]Shanghai Jiao Tong University, [2]University of Michigan, [3]Institute of Computing Technology, CAS

***Abstract*—Numerous studies have proposed hardware architectures to accelerate sparse matrix multiplication, but these approaches often incur substantial area and power overhead, significantly compromising their usage in dense scenarios. On the other hand, systolic arrays deliver high efficiency for dense matrix operations, but their application to sparse matrices remains challenging. An ideal design should process both dense and sparse matrices with high efficiency to satisfy performance and versatility requirements.**

In this paper, we introduce DenSparSA, a balanced systolic array centralized architecture that can execute sparse matrix computations with minimal overhead to original dense matrix computations. DenSparSA supports both single-side and dual-side unstructured sparse matrix multiplications with high efficiency. At the same time, the additional hardware required for managing sparsity is compact and decoupled from the conventional systolic array, allowing for minimal power overhead when switched back to dense matrix operations via circuit gating. The proposed design is implemented with Nangate 45nm. Implementation results show that DenSparSA achieves a speedup ranging from 1.9× to 22× compared to the classic systolic array for sparse workloads, while maintaining relatively low area and power overhead. For dense workloads, the power overhead can be reduced to 12% for BF16 and 5% for FP32. Compared with existing solutions for sparse acceleration, DenSparSA delivers competitive (0.82×-1.32×) efficiency in sparse scenarios and 1.17×-2.28× better efficiency for dense scenarios, indicating a better balance between both situations.

***Index Terms*—Sparsity, Systolic Arrays, Acceleration, Neural Network**

I. Introduction

Systolic arrays have emerged as a leading solution for computing dense matrix operations, exemplified by Google's Tensor Processing Unit (TPU) [1], a state-of-the-art systolic array architecture. The TPU's core matrix multiplication engine employs a 256 × 256 array of identical processing elements (PEs) to perform 8-bit Multiply-and-Accumulate (MAC) operations. In its third-generation design (TPU v3) [2], the architecture features a 128 × 128 array capable of supporting floating-point (FP) arithmetic.

As deep neural networks increasingly adopt sparsity to improve efficiency, there is a growing demand for systolic arrays to reap the benefits from unstructured sparse matrices while maintaining their efficiency in dense matrix multiplication. By providing robust support for both dense and sparse computations, systolic arrays can deliver substantial performance gains for AI accelerators, spanning a wide range of applications from training to inference [3].

To enable systolic arrays for sparse matrix multiplication, researchers have proposed various approaches. However, many of these methods either rely heavily on software-assisted algorithms [3] or introduce significant overheads, such as increased power and area costs, to support unstructured single-sided and dual-sided sparsity. Although these overheads are affordable by the computation savings achieved through sparsity, they remain problematic for dense matrix multiplication, where no such savings are realized. These overheads significantly constrain the use of systolic arrays to sparse matrix multiplication, undermining their original efficiency in processing dense matrix operations. In general-purpose scenarios, such as tensor processing units (TPUs), efficient support for dense matrix multiplication is equally critical.

An Zou is the corresponding author, supported by NSFC 62202287.

This paper presents DenSparSA, a novel architecture and microarchitecture design for systolic arrays aimed at efficiently supporting general sparse matrix multiplication (i.e., including both single-sided and dual-sided unstructured sparse matrix multiplications) while maintaining the efficiency in dense matrix multiplication. The key contributions of this paper can be summarized as follows:

- Unified Support for Unstructured Sparsity: DenSparSA introduces an innovative and efficient systolic array architecture that effectively handles both single-sided and dual-sided unstructured sparsity, with no reliance on fixed sparsity pattern, software-preprocess or external sparsity filters.
- Decoupled Hardware for Efficient Sparsity Management: The design incorporates compact, decoupled hardware modules to efficiently manage sparsity. Hardware support for sparsity could be turned off by clock-gating without interfering with the computation of dense matrix multiplication. Therefore, power overhead for dense workloads are greatly reduced, bringing about better generality in both dense and sparsity scenarios.
- Comprehensive Evaluation and Benchmarking: We implement the proposed DenSparSA using ASIC designs, demonstrating speedups from 1.9× to 22× for sparsity, with relatively low overhead compared to the classic systolic array. When compared to existing solutions in multiple workloads, DenSparSA achieves competitive area efficiency (0.82×-1.32×) and power efficiency (1.17×-2.28×).

The following content is organized as follows: Sec. II presents the background and related work for the systolic array and related approaches for sparse matrix multiplication. Sec. III introduces the methodology incorporating dataflow and architecture designs of DenSparSA. Sec. IV describes the microarchitecture designs to support the data flow. Sec. V discusses the extensive experiments and compares the DenSparSA with the existing solutions. Sec. VI concludes this paper.

II. Background and Related Work

A. Systolic Array

Systolic arrays, also known as Matrix Processing Units (MXUs) or General Matrix Multiplication Accelerators, are specialized hardware architectures designed for parallel dense matrix multiplication, a fundamental operation in deep neural networks [4]. As shown in later Fig. 2, the systolic array consists of a grid of processing elements (PEs) and a three-level memory hierarchy (buffers) to execute computations in parallel, enabling highly efficient parallel processing [5]. Each PE performs multiply-accumulate operations and communicates with adjacent PEs or buffers, making systolic arrays adept at managing input data and weight sharing. They can be scaled to meet computation demands while reducing memory bandwidth

requirements. This efficiency in handling dense matrix multiplication makes systolic arrays integral to accelerating deep neural networks (DNN) tasks [4]. However, while systolic arrays are highly effective for linear operations like dense matrix multiplication (e.g., im2col-based convolution), they cannot efficiently compute sparse matrix multiplications, which are common in compressed DNNs and other heavy computation tasks.

B. Sparse Matrix Multiplication

Matrix multiplication constitutes a major workload in many domains, including the training and inference of DNNs and scientific computation such as graph computation [6, 7] and fluid dynamics [8]. Among these workloads, sparsity is a common scenario, leading to the possibility of extra speedup and improved efficiency by skipping the zero-valued elements that do not contribute to the computation result. A special case of sparsity, structured sparsity, refers to the situation of having predictable patterns of zeroes in matrices, allowing for optimization in memory and computation in deep learning models as predictable sparsity patterns reduce the complexity of hardware support and are approachable by special pruning methods [9]. Different DNN models lead to different scenarios of sparsity. CNN-based models have sparsity in both the weight matrix and activation matrix and sparsity of transformer-based models exists mainly in the weight matrix. The hardware overhead induced by supporting sparsity is often referred to as sparse tax [10, 11], and different strategies in handling sparsity lead to different trade-offs between overall performance and sparse tax.

C. Existing Solutions for Sparse Matrix Multiplication

To reduce hardware costs on random sparsity patterns, many works apply software-hardware co-design to gain speedup from sparsity. Structured sparsity in DNNs is welcomed by both research outcomes like Highlight [11] and commercial products like NVIDIA Ampere [12]. Also, designs like Sense [13] apply a special pruning method on CNN to produce balanced sparsity and reduce the hardware complexity but lack generality for other workloads. However, structured pruning tends to result in larger precision loss and lower sparsity, limiting the effect of speedup [14], and pruning specific to accelerator architecture compromises with the generality of the hardware platform.

Given the limit of structured or specially pruned sparsity, solutions targeting unstructured sparsity emerge. Existing ASIC solutions for unstructured sparse matrix could be divided into two categories: some are based on slightly modified TPU-like systolic array [3, 15, 16] while others are not [17, 18, 19].

STPU [3] and Mentha [15] are two TPU-like designs that adopt data packing to tackle sparse workloads. STPU applies a co-design of packing algorithm and hardware support to leverage sparsity, requiring packing to be done offline. Mentha is also oriented on data packing, but the packing algorithm could be conducted either online or offline. While achieving moderate speedup, STPU and Mentha incur relatively low cost overhead since the modification to a TPU-like baseline is limited. S2 Engine [16], another TPU-like design, does not require data packaging but is limited to dual-side sparsity in CNN. Though having a small cost overhead, designs based on systolic arrays are limited either in performance or in working scenarios.

Aside from systolic arrays, other architectures, on the other hand, achieve higher speedup but induce rather large overhead. SIGMA [17] targets acceleration of irregularly shaped and sparse matrices, leveraging single-side sparsity solely. DSSA [19] leverages dual-side sparsity with a systolic architecture, providing better speedup for CNN workloads than SIGMA while still incurring large overhead. Trapezoid [18] is a solution with high generality, utilizing both single-side and dual-side sparsity for high speedup over a wide range of scenarios. Although these designs provide higher performance for sparse workloads, they involve rather large areas and power overhead, leading to reduced efficiency when handling dense workloads. While numerous previous designs have been proposed to accelerate sparse calculation, they couldn't meet the need to provide a general hardware solution to achieve high performance and efficiency with both sparse and dense workloads.

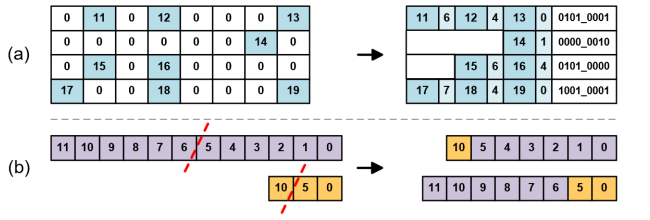


Fig. 1. Examples of dataflow compression in DenSparSA. (a) shows Vector Compression for dual-side sparsity and (b) shows Vector Permutation for single-side sparsity.

III. Dataflow and Architecture Design

In this section, we introduce the dataflow and architecture designs of the proposed DenSparSA.

A. Dataflow Support for Sparsity Computation

For systolic arrays, the total computation latency depends on the time required for data to fully pass through the array. Therefore, the quest for speedup is essentially a quest for shorter data passage time, and acceleration could be made if the hardware support calculation upon compressed dataflow with shorter length and shorter passage time. Based on such principle, a system of compressed dataflow is designed to enable general speedup in different situations of sparsity. The dataflow proposed here would serve as the target for our microarchitecture design.

1) Vector Compression for Dual-side Sparsity:

For dual-side sparsity, both input matrices could be compressed by excluding zero elements, as illustrated in Fig. 1(a). The input matrices that enter the array vertically and horizontally are respectively compressed according to column and row vectors, leading to a shortened length of input on both directions. After compression, each remaining element carries an index that marks its position in the original vector, and a series of elements holds a bit mask representing its sparsity pattern.

However, such a compression method will result in index mismatch when the compressed vectors arrive at the processing element (PE) arrays and lead to incorrect computation results. We will discuss in detail how to design microarchitecture to address this issue in section IV(A).

2) Vector Permutation for Single-Side Sparsity:

In the case of single-side sparsity, only one of the input matrices could be compressed while the other is left complete. This leads to a problem: even though computation between mismatched elements is made possible by certain hardware modification, there is no effect of speedup since the data passage time of the dense input matrix remains unchanged. This leads to restricted usage, since transformer-based DNN models holds only single-side sparsity.

To address this problem, we propose a method called Vector Permutation (VP) to balance the length of compressed sparse vectors

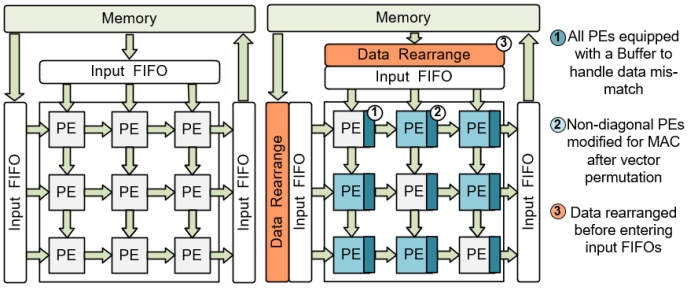


Fig. 2. DenSparSA architecture compared with the conventional systolic array.

and uncompressed dense vectors. As is shown in Fig. 1(b), each input vector is cut into two halves by the median index, after which the later halves are exchanged between each pair of vectors symmetric about the diagonal. The vector length after permutation is roughly the average of the original length of the two paired vectors. Please note that the ascending sequence of the element indices are maintained after permutation, which plays a key role in DenSparSA's approach to accelerating with single-side sparsity..

For single-side sparsity, VP is a must in our design to achieve speedup. For dual-side sparsity, VP could also provide the additional effect of speedup by balancing the length of inputs on two sides. The hardware implementation of vector permutation and computation details of permuted vectors will be discussed in section IV(B).

3) Vector Filter for Extra Speedup:

In matrix multiplication scenarios such as DNN, a weight matrix is pre-known. If the pre-known weight matrix has considerable sparsity, such sparsity could be utilized ahead of computation to achieve extra speedup. In such a case, DenSparSA acquires a merged sparse vector out of a number of vectors in the pre-known sparse weight, after which the other input matrix is filtered by this merged sparse vector to shorten its length before computation.

Such an approach brings about better speedup but results in array partitioning and corresponding cost overhead, the details of which will be discussed in section IV(C).

B. Architecture Support for Proposed Dataflow

To support the proposed dataflow, the new hardware architecture should be able to 1) perform Multiply-and-Accumulate (MAC) operations between compressed vectors with mismatched data, 2) perform MAC between permuted vectors and 3) rearrange the data inputs to realize vector permutation and filtering. In order to fulfill the requirements as listed above, we make a series of modifications to a classic systolic array which has the same architecture as the matrix multiplication unit in the Tensor Processing Unit (TPU), and propose a new architecture based on it, as illustrated in Fig. 2.

The fundamental concept behind this design comprises mainly three essential microarchitecture improvements: ① All PEs are equipped with a buffer to handle data mismatch. ② Non-diagonal PEs are slightly modified to support VP, while buffered PEs on the diagonal are naturally adaptable to VP. ③ Data are rearranged before entering the input FIFOs in order to implement VP and filtering. The above modifications, especially ① and ② which contribute to the major part of hardware overhead, are carefully designed to reduce their impact on the inherent structure of a classic (TPU-like) systolic array. The operations of the PEs and buffers are pipelined to minimize changes to the critical path. The buffers are compact and can be decoupled from the array, allowing them to be clock-gated when switching back to dense matrix multiplication. As a result, support for sparsity is achieved with reduced impact on the area and power efficiency of the original systolic array during dense workload execution.

IV. Microarchitecture Design and Optimization

This section will dive into the details of the microarchitecture, explaining the design and operation of hardware modules required to support previously proposed dataflow.

A. Buffered PE for Index Matching

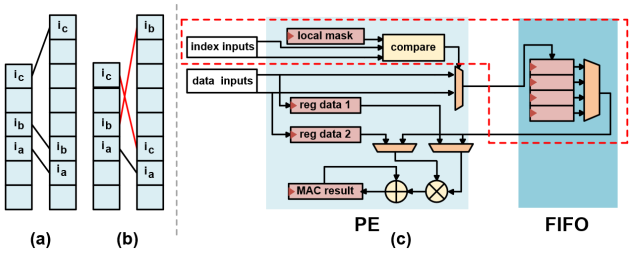


Fig. 3. Dataflow (a, b) and hardware structure (c) of buffered PE. All components in (c) marked by a reg triangle are clock-triggered.

Data in compressed vectors are mismatched when they arrive at a processing element (PE), of which an archetypal situation is shown in Fig. 3(a) with i_x denoting the ascending indices of corresponding elements. One element of a matching pair arrives at the PE earlier than another, for which extra storage is needed to compute multiplication correctly. Also, the intersection between matching pairs as illustrated in Fig. 3(b) is impossible as it conflicts with the ascending order followed by indices in the same vector. Therefore, extra storage needed to handle data mismatch could be implemented in the form of a FIFO buffer. The interact logic between a PE and its buffer is shown in Algorithm 1. The PE holds a local bit mask denoting the indices of matching pairs, by which it could tell whether an element should be preserved for future calculation. Values in the leading one of the matching pairs are pushed, and a value is popped from the buffer when the later one arrives.

Algorithm 1: Access Logic of Buffer

Input: $index_A, data_A, index_B, data_B$
begin
  if $index_A > index_B$ **then**
    if $index_A$ *found in mask* **then**
      push $data_A$
    if $index_B$ *found in mask* **then**
      pop
  if $index_B > index_A$ **then**
    if $index_B$ *found in mask* **then**
      push $data_A$
    if $index_A$ *found in mask* **then**
      pop

Apparently, conducting buffer access and Multiply-and-Accumulate (MAC) sequentially in the same cycle would result in a significantly longer critical path, thus buffer access and MAC are pipelined in our design. Conventionally, a PE in a systolic array does calculations upon data received at the input ports. In DenSparSA, the calculation is done upon data transferred to neighbor PEs, i.e., the data at the output ports. Thus, the inputs to the multiplier is the data that arrived a cycle ago. If the two input indices match

naturally, MAC upon the two input values are done in the next cycle. If the latter one of a matching pair arrives, a pop signal is sent to the buffer, and MAC is done between popped data and one of the data passed to a neighbor PE. The pipeline logic is described in Algorithm 2.

Algorithm 2: Pipeline Logic of Buffered PE

Input: $index_A, data-in_A, index_B, data-in_B$
Output: $data-out_A, data-out_B$
begin
  if $index_A = index_B$ **then**
    do MAC on data outputs in the next cycle
  if $index_A > index_B$ & $index_B$ *found in mask* **then**
    do MAC on popped data and $data-out_B$
  if $index_B > index_A$ & $index_A$ *found in mask* **then**
    do MAC on popped data and $data-out_A$

The hardware structure of a buffered PE is shown in Fig. 3(c), where trivial details such as connections with other PEs are eliminated for simplicity. The combinational and sequential logic in the buffer contributes to the major part of area and power overhead. A merit of our design is that PEs are decoupled with their buffers when performing dense matrix multiplication, during which turning the buffer off by clock gating would not affect the function of the PE. When the compute array is switched to dense workloads, the clock connected to components framed in a red dotted line in Fig. 3(c) is gated, and the dynamic power due to index propagation, local bit mask refreshment, index comparing and FIFO will not contribute to power overhead.

FIFO depth is a crucial parameter in our design. When a FIFO is full, the operation on the corresponding PE has to stall, leading to a slowdown in computation. Thus, large FIFO depth reduces the overall stall rate but results in larger hardware overhead. On the other hand, FIFOs constitutes the main part of overhead in our design. Though they do not account for significant chip area, they lead to larger power overhead. Therefore, a compromise should be made between hardware overhead and lower stall rate. Since the stall rate is hard to model mathematically, we apply the Monte Carlo method to simulate the overall stall ratio when FIFOs are configured with different depths. This is done by feeding random sparsity patterns with varying level and combination of sparsity to a self-built cycle-accurate simulator, and an expectation of stall rate is calculated from the simulation outcomes. The result is shown in Table. I. In our implemented design, the FIFO depth is chosen to be 6 and the exact stall rate is 8.61×10^{-4}.

TABLE I
RELATION BETWEEN STALL RATE AND FIFO DEPTH.

FIFO depth	0	1	2	3	4
Stall rate	0.998	0.724	0.323	0.109	0.028
FIFO depth	5	6	7	8	9
Stall rate	0.0056	0.000861	3×10^{-6}	2×10^{-8}	0

B. Calculation upon Permuted Vectors

As is discussed in III, Vector Permutation (VP) promotes speedup by balancing the length of inputs. The buffered PEs described in V-C could be easily adapted to MAC between permuted vectors. We'll begin with the situation of a single PE, and then extend the design to a whole PE array.

In VP, original vectors are cut according to the middle index of the longer vector. Therefore, only the part with larger indices than that middle index are exchanged during permutation of the later halves of the vectors. Such a process maintains the ascending order of the indices in each vector, as could be shown in Fig. 1(b). On the other hand, the logic of FIFO-buffered PE is applicable for inner products of matched elements as long as the indices are ascending in each vector. Consequently, an isolated buffered PE naturally supports VP.

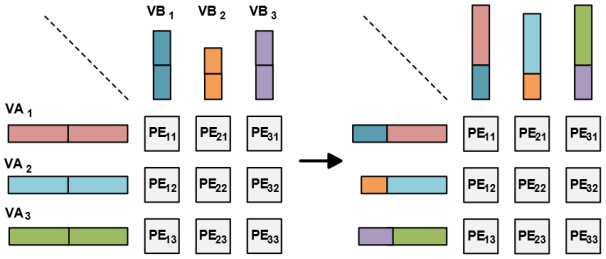


Fig. 4. Showcase of Vector Permutation for a PE array.

The situation is a bit different for an array of PEs. In this case, each side of the array receives a batch of vectors, and vector permutation is performed between each pair of vectors symmetric about the diagonal, as illustrated in Fig. 4. For the PEs on the diagonal, the situation is the same as compared to an isolated PE, since the former and later halves contribute to inner products between the same pair of original vectors. For non-diagonal PEs, however, the former and later halves contribute to inner products between different vector pairs.

Assume the longer vectors have length $2n$ or $2n+1$. For PE_{ij}, the MAC result in the first n cycles accounts for vector_{Ai} and vector_{Bj}, while MAC result in later cycles accounts for vector_{Aj} and vector_{Bi}. An exactly opposite situation holds true for PE_{ji}, which is symmetric to PE_{ij} about the diagonal. Thus, by exchanging the MAC result between PE_{ij} and PE_{ji} after the nth cycle for once, each of the pair of symmetric PE gets the correct MAC result between a certain pair of permuted vectors at the end. The hardware cost for this is just a MUX for each non-diagonal PE to decide whether the MAC result should be refreshed based on the result from the same PE or another PE.

C. Data Rearrange

Modules for data rearrangement are introduced to conduct VP and filtering. In a classic design of a systolic array, input data are held by double-buffered FIFOs before entering the compute array. In our design, a filter and a router are inserted at the interface of each input FIFO and array memory. The detailed microarchitecture is illustrated in Fig. 5. The bit masks of a group of sparse vectors are merged by OR operation, producing a merged mask to be used as a filter for the denser input matrix. The index of each element from the dense matrix is compared with the merged bit mask, and only elements with a matching index can enter the input FIFO. Filtering is only conducted for the denser side of the inputs since the benefit of filtering an already compressed sparse input is limited as compared to its overhead.

The router is a combination of two 2-1 MUXs, determining which source should the input FIFOs receive data from. Dense input vectors filtered by the same mask share the same series of indices and consequently the same middle index, therefore the flag signal of MUXs could be shared between them, reducing the complexity of control. Routing of this form is applied between each pair of input vectors symmetric about the diagonal of the array.

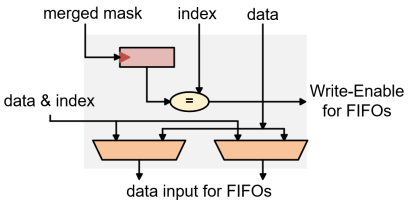


Fig. 5. Data rearrange at array-memory interface.

With filtering comes the need to partition the compute array, since dense inputs filtered by a certain group of sparse vectors are no longer applicable to another group of sparse vectors. The size m of a sparse vector group to be merged into a mask is positively correlated with overheads of partitioning as m is exactly the partition granularity, and negatively correlated with the speedup effect as smaller m brings about higher sparsity in the merged mask. As for our design where medium sparsity constitutes the major workload, m is chosen to be 4 to make a balanced compromise between overhead and speedup.

V. Implementation and Evaluation

A. Implementation

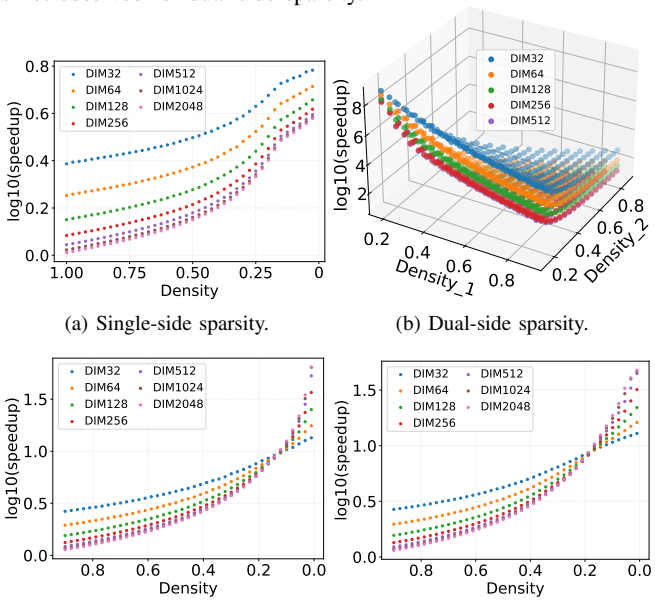
We implemented the proposed DenSparSA design and the baseline systolic array design, originally used by Tensor Processing Units (TPUs), in RTL. The designs were synthesized at 400MHz using OpenROAD and Innovus with the NanGate 45-nm open-cell library. We assume 128×128 PE configuration across all designs. In DenSparSA, each processing element (PE) includes a FIFO with a depth of 6, and the array works by 4×4 partition for Vector Filtering. The multiply-accumulate (MAC) units are designed to support BF16 or FP32 multiplication and accumulation. In the following, we compare the proposed DenSparSA with the original systolic array (SA) and related state-of-the-art approaches.

B. Performance and Cost Breakdown

1) Speedup Performance:

We evaluate the performance of DenSparSA in terms of speedup over a conventional TPU-like systolic array. Matrix with dimensions ranging from 32 to 2048 are taken into account. The relation between speedup and dual-side sparsity is shown in Fig. 6, with speedup ratios expressed in log forms. The co-effects of sparsity on both sides are shown in Fig. 6(b). For the quantitative showcase, Fig. 6(c) and Fig. 6(d) show the cases of having equal sparsity on two sides and having sparsity x on one side with sparsity range $[x - 0.2, x + 0.2]$ on another side. For sparsity ranging from 50% to 95%, DenSparSA achieves 1.9× - 5.5× gmean speedup with single-side sparsity and 2.6× - 17.5× gmean speedup with dual-side sparsity, as compared with baseline design of the systolic array.

As can be seen in both figures, there's a speedup for matrices with smaller dimensions even when the sparsity is very low. This is due to the effect of solely partitioning since a smaller array size brings about lower loading latency, a side effect of our partitioning that is intended for filtering vectors. With the decrease in matrix density, the speedup effect converges to the case of matrices with larger dimensions, as the advantage brought about by just partitioning becomes trivial. For single-side sparsity displayed in Fig. 6(a), the growing trend of speedup rate is retarded as sparsity grows high. This is because the denser one of the two matrices has to be compressed by filters to achieve higher speedup, and when the sparsity is high, the time cost for matrix compression exceeds the time cost for computing the compressed matrix, since the former time cost is fixed given the length of the dense input matrix and the latter one goes down when the sparsity of another input matrix is very high. Therefore, there is a saturation on the overall speedup effect. Allocating more hardware resources for data compression could help with the saturation, but would not be cost-effective for DNN-level sparsity. Such saturation is not observed for dual-side sparsity.



(a) Single-side sparsity. (b) Dual-side sparsity.

(c) Dual-side with equal sparsity. (d) Dual-side with varying sparsity.

Fig. 6. Speedup under different situations of sparsity.

2) Cost and Overhead Breakdown:

Table II shows the area breakdown of our design. The analysis of area usage is focused on the computing array since no modification is made to the structure of cache and external memory. The total overhead is 15% for BF16 multiplier and 12% for FP32 multiplier. The hardware overhead mainly comes from three sources: additional buffers, additional control logic in PEs, and data rearrangement, as reflected in the increased area in PEs and input modules. Although each PE gets additional logic for buffering, the overall area overhead is limited, since sequential cells account for a rather small proportion of the whole area, and the additional control logic is not complex. The overhead of buffered PE is further mitigated as larger multipliers are applied, since the growth of hardware cost for buffers, i.e., MUXs, comparators, and registers are in a roughly linear relation with data width, making it hard to catch up with the growing hardware cost of multipliers.

TABLE II
Area Breakdown Compared with Baseline Design.

Breakdown (mm^2)		**BF16**				**FP32**			
		TPU-SA		DenSparSA		TPU-SA		DenSparSA	
Array	Reg	102.1	14.3	116.4	25.2	186.7	27.4	207.5	36.6
	MAC		78.6		78.8		136.2		139.5
	Other		9.2		12.5		23.1		31.4
Input		1.5		2.8		2.8		4.8	
Output		1.7		1.7		3.2		3.2	
Other		6.2		7.5		10.8		12.96	
Total		111.5		128.4(+15%)		203.5		228.5(+12%)	

Table III shows the power breakdown of DenSparSA. Still, buffered PE contributes to the majority of the overhead, and the power overhead is more significant than the area overhead since buffers account for a larger proportion of power consumption. The overhead

caused by buffers is partly diluted by the stage registers in multipliers pipelined to achieve higher clock frequency but is still a drawback if no benefit could be made from accelerating the workload, in other words, if the workload is dense.

Unlike area statistics, the total power is reported respectively in sparse mode and dense modes. In sparse mode, routers and buffers of the PEs are turned on to facilitate computation on sparse matrices. In dense mode, unnecessary components as discussed in Section V-C are turned off by cutting off the dynamic power consumed for handling sparsity. Decoupling is quite effective in reducing power overhead for dense workloads. By clock-gating, power overhead in dense workloads is reduced to 12% and 5% for BF16 and FP32 multipliers. The feature of decoupled sparsity support put our design at an advantage when switched back to a dense workload, which will be further discussed in Section V-C through comparison.

TABLE III
POWER BREAKDOWN COMPARED WITH BASELINE DESIGN.

Breakdown (W)	BF16			FP32		
	TPU-SA	DenSparSA		TPU-SA	DenSparSA	
		Sparse	Dense		Sparse	Dense
Array	13.7	22.9(+52%)	14.4(+6%)	50.5	61.4(+21%)	52.0(+3%)
Input	2.4	3.7		4.7	7.0	
Output	0.8	0.8		1.3	1.3	
Other	0.6	0.7		0.9	1.0	
Total	17.5	26.1(+49%)	19.6(+12%)	57.6	71.2(+24%)	61.2(+5%)

C. Comparison among Existing Solutions

We compare DenSparSA comprehensively with existing solutions for accelerating sparse matrix calculation. The comparison is based on two principles. First, comparison is done one-by-one with each design according to the datatype and evaluation metrics proposed by each respective paper. This is because evaluation metrics vary from accelerating DNN workloads to solely accelerating different matrix benchmarks, and we lack information to derive each design's performance for a generalized theme. Second, statistics of performance, power, and area are normalized to a baseline design of a TPU-like systolic array. This is because even basic statistics such as the cost of a TPU-like baseline PE mismatch among different works, and a consistent comparison of absolute values between each work would be hard to conduct given such mismatch. Therefore, the speedup and overhead ratios over a TPU-like baseline design, as reported in all compared works, are chosen as a common measure for reference.

Fig. 7 shows the comparison of computing efficiency in sparse scenarios. Area efficiency, as displayed in Fig. 7(a), is calculated as speedup ratio over baseline TPU divided by area overhead, and the same is done for power efficiency. Workloads include dual-side dense (DD), single-side sparse (SD), and dual-side sparse matrix (SS) multiplication with sparsity on the order of 10^{-1}, the common sparsity for DNN-related computation, and a geometric mean is calculated for designs capable of both SS and SD workloads. As for the displayed bar chart, the efficiency is normalized to the better design between each comparison pair, with the original values of $(Perf/Area)$ and $(Perf/Power)$ marked on top of each bar. DenSparSA provides better area efficiency than SIGMA, DSSA and Mentha, and is slightly lower than Trapezoid. As for power efficiency, DenSparSA is 24% and 32% ahead of SIGMA and Mentha, and 18% and 7% less efficient than DSSA and Trapezoid. From the comparison above, we can conclude that DenSparSA delivers a competitive (0.82×-1.32×) computing efficiency among existing sparse matrix accelerators.

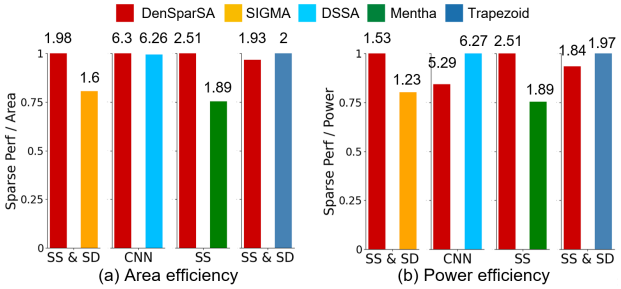


Fig. 7. Comparison with Existing Solutions in Sparse Scenarios

Figure 8 shows the comparison of computing efficiency in dense scenarios. For dense workloads, DenSparSA delivers the best (1.17×-2.28×) computing efficiency in terms of both area utilization and power consumption. The advantage in area utilization is due to the originally small area overhead, as the extra logic is quite compact and buffers do not account for a large proportion of chip area. On the other hand, the advantage of DenSparSA in power efficiency is largely due to the "pluggable" feature of decoupled modification upon baseline design. Since a major contributor of power overhead could be turned off, the power efficiency is rather close to the baseline model. In our evaluation, only clock gating for buffered PEs is modeled and simulated, which already provides a meaningful drop in power overhead. DSSA and Trapezoid provide slightly higher computing efficiency over DenSparSA in sparse workloads, but are outperformed by DenSparSA by a larger gap in dense workloads, indicating that DenSparSA may hold better versatility in scenarios where dense and sparse workloads are mixed together, and that DenSparSA maintains a better balance between leveraging sparsity acceleration and keeping efficient in conventional situations.

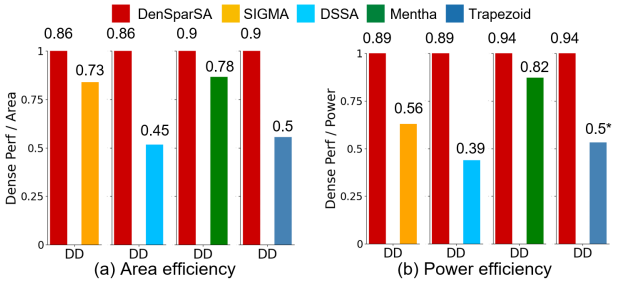


Fig. 8. Comparison with Existing Solutions in Dense Scenarios

VI. CONCLUSION

This paper presents DenSparSA, a systolic array centralized architecture that leverages general sparsity for speedup in matrix multiplication. The dataflow and architecture are proposed to shorten the data passage time and bring about speedup benefits, for which a series of hardware modifications are made upon the classic design of a 2D systolic array. The microarchitecture and circuit support for sparsity handling is compact and decoupled from the original design of the systolic array, therefore clock-gating could be applied to minimize power overhead when switched back to dense workloads. Compared with existing solutions, our design achieves competitive (0.82× - 1.32×) performance in general sparse matrix multiplication while holding a better balance between dense and sparse workloads with better (1.17× - 2.28×) computing efficiency for dense scenarios, maintaining the benefit from high power efficiency of original systolic arrays in dense scenarios.

REFERENCES

[1] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[2] Norman P Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. A domain-specific supercomputer for training deep neural networks. *Communications of the ACM*, 63(7):67–78, 2020.

[3] Xin He, Subhankar Pal, Aporva Amarnath, Siying Feng, Dong-Hyeon Park, Austin Rovinski, Haojie Ye, Yuhan Chen, Ronald Dreslinski, and Trevor Mudge. Sparse-tpu: Adapting systolic arrays for sparse matrices. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12, 2020.

[4] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. In *Annual Design Automation Conference*, 2017.

[5] Jie Wang, Licheng Guo, and Jason Cong. Autosa: A polyhedral compiler for high-performance systolic arrays on fpga. In *FPGA*, 2021.

[6] Jeremy Kepner, David Bader, Aydın Buluç, John Gilbert, Timothy Mattson, and Henning Meyerhenke. Graphs, matrices, and the graphblas: Seven good reasons. *Procedia Computer Science*, 51:2453–2462, 2015.

[7] David Emms, Edwin R Hancock, Simone Severini, and Richard C Wilson. A matrix representation of graphs and its spectrum as a graph invariant. *arXiv preprint quant-ph/0505026*, 2005.

[8] EI Yakubovich and DA Zenkovich. Matrix approach to lagrangian fluid dynamics. *Journal of Fluid Mechanics*, 443:167–196, 2001.

[9] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[10] Hyeonuk Sim, Jooyeon Choi, and Jongeun Lee. Spartann: Sparse training accelerator for neural networks with threshold-based sparsification. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 211–216, 2020.

[11] Yannan Nellie Wu, Po-An Tsai, Saurav Muralidharan, Angshuman Parashar, Vivienne Sze, and Joel Emer. Highlight: Efficient and flexible dnn acceleration with hierarchical structured sparsity. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1106—-1120, 2023.

[12] Jack Choquette, Edward Lee, Ronny Krashinsky, Vishnu Balan, and Brucek Khailany. 3.2 the a100 datacenter gpu and ampere architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 48–50, 2021.

[13] Wenhao Sun, Deng Liu, Zhiwei Zou, Wendi Sun, Song Chen, and Yi Kang. Sense: Model-hardware codesign for accelerating sparse cnns on systolic arrays. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 470–483, 2023.

[14] Ziheng Wang. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 31–42, 2020.

[15] Minjin Tang, Mei Wen, Yasong Cao, Junzhong Shen, Jianchao Yang, Jiawei Fei, Yang Guo, and Sheng Liu. Mentha: Enabling sparse-packing computation on systolic arrays. In *Proceedings of the 51st International Conference on Parallel Processing*, pages 1–11, 2022.

[16] Jianlei Yang, Wenzhi Fu, Xingzhou Cheng, Xucheng Ye, Pengcheng Dai, and Weisheng Zhao. S 2 engine: A novel systolic architecture for sparse convolutional neural networks. *IEEE Transactions on Computers*, 71(6):1440–1452, 2021.

[17] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella1, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *IEEE Symposium on High-Performance Computer Architecture (HPCA)*, pages 58–70, 2020.

[18] Yifan Yang, Joel S. Emer, and Daniel Sanchez. Trapezoid: A versatile accelerator for dense and sparse matrix multiplications. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 931–945, 2024.

[19] Zhengbo Chen, Qi Yu, Fang Zheng, Feng Guo, and Zuoning Chen. Dssa: Dual-side sparse systolic array architecture for accelerating convolutional neural network training. In *Proceedings of the 51st International Conference on Parallel Processing (ICPP)*, pages 1–10, 2022.

SuperFast: Fast Supernet Training Using Initial Knowledge

Moritz Thoma[1,3†*] Emad Aghajanzadeh[1,3†] Shambhavi Balamuthu Sampath[1,3]
Pierpaolo Mori[3] Nael Fasfous[3] Alexander Frickenstein[3] Manoj-Rohit Vemparala[3]
Daniel Mueller-Gritschneder[2] Ulf Schlichtmann[1]

[1]Technical University of Munich [2]TU Wien [3]BMW Group
† Equal Contribution. *Corresponding author moritz.thoma@tum.de.

***Abstract*—Once-for-all based neural architecture search (NAS) proposes to train a supernet once and extract specialized subnets from it for efficient deployment. This decoupling between training and search enables easy multi-target deployment without retraining. Nevertheless, the initial training cost has remained extremely high, with SOTA approaches like ElasticViT and NASViT taking more than 72 and 83 GPU days respectively. While other approaches have tried to accelerate the training by warming up the largest model in the search space, we argue that this is suboptimal, and knowledge is easier scaled upward than downward. Hence, we propose *SuperFast*, a simple, plug and play workflow, that (I.) pretrains a subnet of the supernet search space, and (II.) distributes its knowledge within the supernet before the training. SuperFast offers a substantial acceleration in the supernet training, resulting in a significantly better accuracy vs. training-cost trade-off. Using SuperFast on both ElasticViT and NASViT supernets achieves the baseline's accuracy 1.4× and 1.8× faster on the ImageNet dataset. Moreover, for a given time budget, SuperFast improves accuracy vs. latency trade-offs for subnets, gaining 4.0 p.p. for the 20-50 ms range on Pixel 6. Code available in https://github.com/MoritzTho/SuperFast.**

I. Introduction

The success of deep neural networks (DNN) models in various challenging tasks makes them attractive for use in real-world applications. However, the increasing growth in network's complexity and size makes their deployment challenging under mobile settings. On the other hand, bringing models to real world applications often requires deploying them on a wide range of different hardware devices. This is challenging as devices have different compute, bandwidth and power resources thereby resulting in different optimal architectures for different devices.

Neural architecture search (NAS) has explored many different ways to optimize the model architecture for any given hardware. However, as the number of different deployment target devices grows due to generational improvements, multiple performance tiers or switch of suppliers, so does the effort to find optimal architectures for each of them, growing the neural network design cost linearly with each new target (blue line in Fig. 1). To avoid a linear training cost increase with growing number of deployment targets, [1] introduced a once-for-all approach that creates one large supernet that is trained once and subsequently specialized for specific deployments, thereby decoupling the training and the search stage. Importantly, the single training mechanism for the supernet jointly optimizes all subarchitectures in the supernet, consequently not requiring finetuning of the specialized architectures extracted after the joint training finished. As a result, the training cost stays the same, even when the number of deployment targets grows (yellow line in Fig. 1).

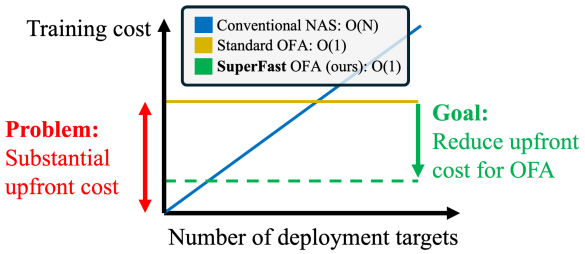


Fig. 1: For conventional NAS training cost increases linearly with the number of targets. Supernet NAS approaches reduce this to a single upfront training cost that pays off when targeting many devices. Our SuperFast method reduces this extremely high upfront cost, making supernet NAS more accessible.

Subsequent works have improved task performance by upgrading the search space architectures [5], [9], improving the sampling of architectures during joint training [9], [11] and improving the way knowledge is distributed between subnets during training [10], [11]. However, despite all these improvements, the training time has still remained a substantial concern in state-of-the-art taking more than 1750h [9], 2000h [5] on a NVIDIA A100 80G, making the training of supernet prohibitively expensive for most use cases. While some works have made attempts to reduce the training time of the supernet, they impose severe search space restrictions [7] or require warm up of the supernet [8], which we find is both inefficient and ineffective.

Differently, in this work we aim to reduce the expensive training time, using knowledge from a single, *smaller* architecture within the search space to infuse the entire search space with initial knowledge before starting the training. This simple, yet powerful approach allows for faster training and higher accuracies, thereby significantly reducing the GPU days required to achieve high performance.

The three main contributions of this paper can be summarized as follows:

- We propose SuperFast, the first method to make use of a small pretrained architecture to substantially accelerate the training of a large supernet search space.
- We show that using a small pretrained architecture together with our upscaling paradigm is more effective and time efficient than previous approaches relying on warmed up or fully trained supernets.
- Finally, we demonstrate the efficacy of our simple, yet effective SuperFast method on both ElasticViT and NASViT search spaces. We achieve a significant 1.4× and 1.8× time reduction while meeting the same accuracy as the baseline. We further show substantial accuracy improvements of up to 4.0 p.p. at a range of Pixel 6 latency budgets on the ImageNet dataset over the standard supernet training.

II. Related Work

A. Once-for-all Training

Early NAS methods have achieved significant advancements. However, their need to repeat search and training for each new hardware target makes them prohibitively costly in multi-target deployments. To address this issue, OFA [1] proposed a two-stage NAS to decouple training and search phases. OFA trains one large supernet once, and subsequently searches subnet architectures, that are specialized for different deployment targets. After the search, the specialized subnets only need minor fine tuning, leaving only one substantial training cost.

1) Updated Search Space: Followed by recent the success of vision transformer (ViT) models, hybrid CNN-ViT models have gained popularity for various vision tasks. NASViT [5] upgrades the OFA supernet architecture to such a SOTA hybrid CNN-ViT architecture, and for the first time, outperform prior pure CNN variants on mobile settings. Similarly, ElasticViT [9] uses a hybrid CNN-ViT supernet architecture but replaces the transformer blocks from NASViT with their own, more hardware friendly design. With their design and training method, they represent the current state-of-the-art for accuracy-latency trade-offs in supernets. Given the success of these hybrid architectures, we follow the same supernet search space and training recipes as NASViT and ElasticViT.

2) Training Cost Inefficiency: One challenge of supernet training is the vastness of the search space, including $O(10^{19})$ sub-architectures [1], which makes it challenging to optimize, resulting in prohibitive training cost. The work in [7] proposes to accelerate the optimization of the supernet by correlating different design decisions, reducing the search space to only 243 architectures. This achieves 2× reduction in training cost, but severely limits the flexibility of the resulting supernet, limiting its applicability for searching diverse, hardware specific subnets. BigNAS [11] proposes an in-place knowledge distillation to supervise sampled subnets with predictions made by the biggest model (supernet). This removes the need for further fine-tuning sampled subnets and reduce the training cost and is adopted by most subsequent works including NASViT and ElasticViT. Nevertheless, the finetuning done by OFA after the training is relatively inexpensive. Hence, the overall cost still remains high. InstaTune [8] is the closest to our work. They convert a *fully* trained architecture into a supernet by allowing elasticity to smaller variants of itself. Hence, they initialize their supernet search space with fully optimized weights of the *largest* architecture. Afterwards, they perform supernet training to jointly adopt it to downstream tasks. Differently, we argue that starting with the biggest network as initialization is suboptimal as scaling features down is harder than scaling features up. Hence, our SuperFast method starts with a *medium* sized subnet that *scales up* its knowledge to the supernet.

B. Knowledge Transfer

Knowledge transfer in neural networks seeks effective methods for transferring knowledge of a trained network to another. Knowledge distillation is one of its well-studied subfields to perform the transfer to smaller networks. Inversely, Net2Net [3] offers a rapid knowledge transfer to bigger networks based on network's function-preserving transformations. FNA [4] further improves these transformations, extends them to kernel dimension, and makes the transfer possible in both-directions. Inspired by these works, SuperFast applies an upscaling paradigm to distribute knowledge of a medium sized subnet, to a vast supernet search space. Table I provides a summary comparing our work to discussed related works.

TABLE I: Comparison of our SuperFast approach to related works for faster supernet optimization and knowledge transfer.

Works	Full Search Space	Avoids Supernet Warm Start	Knowledge Scaling
CompOFA [7]	✗	✓	✗
InstaTune [8]	✓	✗	✗
FNA [4]	-	-	✓
SuperFast (ours)	✓	✓	✓

III. Methodology

A. Supernet Background

We define supernet $\mathcal{M}$ with weights $\mathcal{W}$, and the set of containing architectures $\mathcal{A}$ (subnets). The set $\mathcal{A}$ is determined by definition of supernet's search space that creates the elasticity feature of it. Consider $\mathcal{M}$ with a set of micro-architectural blocks as B_i which can be either CNN- or ViT-based. Similar to the previous works, we consider three search dimensions, namely, width, depth, and kernel size. Width refers to the number of convolution channels for CNN and size of hidden feature in ViT blocks. Depth shows the number of layers in each CNN or ViT blocks, and kernel size determines the size of kernel used in CNN blocks. Therefore, each of these blocks is configurable by its depth, width, and kernel size, denoted as D_i, W_i and K_i, respectively.

The overall objective of supernet training can be defined as follows:

$$\min_{\mathcal{W}} \sum_{i \in \mathcal{A}} \mathcal{L}_{\text{val}}(\mathcal{W}_i) \tag{1}$$

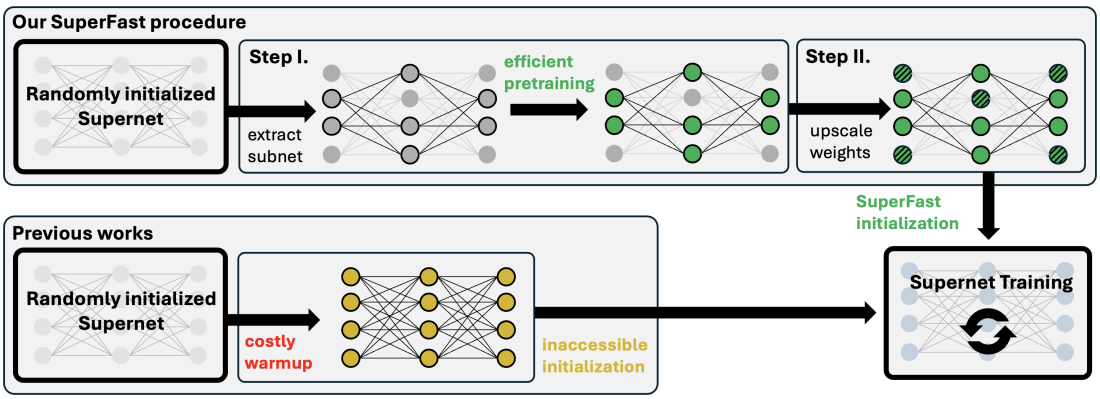


Fig. 2: Overview of our SuperFast procedure. Starting from a randomly initialized supernet, we extract and pretrain a subnet (step I.). Then, we upscale the subnet weights to the size of the supernet, achieving a better initialization more efficiently (step II.). Different to previous works, we do not rely on costly supernet warmup, or even complete supernet training, that may leave key features inaccessible to most subnets.

where $\mathcal{W}_i$ represents weights associated with subnet i and derived from the supernet weight $\mathcal{W}$. As all weights are derived from the single supernet weights $\mathcal{W}$, weights are shared within subnets of the supernet. Hence, supernet $\mathcal{M}$, which by definition is the biggest subnet in $\mathcal{A}$, contains all smaller subnet configurations in its weights $\mathcal{W}$.

To optimize the above objective, a small number of subnet samples (typically 4) are selected in each training iteration. Forward and backward passes are then performed to update the shared weights based on the resulting gradients. The sampling strategy typically follows the sandwich rule, selecting the largest and smallest subnets along with a few random ones in between. The largest subnet (supernet) is trained with labels, while the others use the supernet's predictions as soft labels. This training schema enables the co-optimization of multiple subnets simultaneously. However, it significantly increases the training cost compared to single-model training due to the need for multiple forward and backward steps in each iteration.

B. Primary Objective

The primary goal of this work is to reduce the training time in supernet NAS approaches T_{sup}. For that, we propose SuperFast a simple, plug and play workflow that leverages initial knowledge to accelerate the supernet training, resulting in shorter overall training time.

Specifically, SuperFast (I.) pretrains a single subnet $\mathcal{A}_{init}$ of the supernet $\mathcal{M}$ for time T_{init}, and (II.) distributes its knowledge within the supernet $\mathcal{M}$ before the training, resulting in an enhanced supernet $\mathcal{M}^*$ with a reduced training time T^*_{sup}. The relation is shown in Eq. 2 and the overall workflow is visualized in Fig. 2.

$$T_{\text{init}} + T^*_{\text{sup}} \ll T_{\text{sup}} \tag{2}$$

C. Subnet Selection for Pretraining

As for SuperFast's **first** stage, a subnet is selected from $\mathcal{A}$, denoted as $\mathcal{A}^{pre}_{init}$. The subnet is then trained standalone to achieve strong performance and acquire a sufficient amount of knowledge for effectively performing the task. We opt for pretraining a standalone network as it can acquire initial knowledge much quicker than a complex supernet. Specifically, training a medium sized subnet in the ElasticViT search space on ImageNet for one epoch is more than $8\times$ faster than training the corresponding supernet for one epoch, accelerating the initial knowledge gain substantially. To determine the best size of the network to pretrain, we study the weight sharing mechanism of supernets. There, the weight sharing is dependent on weight position rather than weight importance. Therefore, during pretraining, large networks may learn important features in weights that are inaccessible to most subnets. This problem is highlighted in Fig. 3, where a large subnet configuration has learned information in the upper channels during pretraining that cannot be accessed by smaller networks. Conversely, for smaller networks, the important features are learned in weights, that are shared between most subnets. Based on this insight, we further argue that choosing a small or medium sized subnet results in better knowledge transfer from a standalone, pretrained model to the supernet. This claim is controversial to the common supernet warming-up phase used in prior works such as in OFA [1], CompOFA [7], and NASViT [5].

D. Proposed Knowledge Distribution

The **second** stage leverages the knowledge from $\mathcal{A}^{pre}_{init}$ to provide a strong initialization for the joint optimization problem presented in Eq. 1. This is a challenging task that requires careful consideration, as weight-sharing and joint optimization could gradually diminish the knowledge previously acquired by the subnet. The key component of our approach is the effective knowledge transfer between the pretrained subnet and subnets in $\mathcal{A}$.

The knowledge transfer process varies depending on whether the target network is larger or smaller than $\mathcal{A}^{pre}_{init}$. Therefore, we address each case separately in our discussion: transferring knowledge to larger networks (upwards) and to smaller networks (downwards).

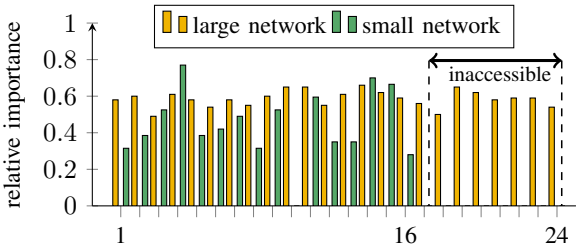


Fig. 3: Relative importance scores of channels within the same layer of two independently pretrained subnets of a supernet. The larger variant uses the 24 channel configuration of that layer and has important weights in the high channels 16-24, that will be inaccessible to smaller subnets that use the 16 channel configuration, degrading the transferability.

1) Downward Transfer: For downward transfer, we make use of the weight-sharing property. More specifically, we have made the following decisions across three search dimensions, i.e., kernel size, channel size, and network depth.

- **Kernel Cropping:** Larger kernels are center-cropped to match the desired size. For example, a 5x5 kernel is reduced to a 3x3 kernel by reusing the central part.
- **Channel Slicing:** The desired number of channels is selected by slicing from the beginning of the channel array. For instance, if 8 channels are needed from a 16-channel layer, the first 8 channels are selected.
- **Layer Utilization:** To reduce network depth, only the first few layers of the network are used. Specifically, the first L_i layers are utilized instead of the full set of N_i layers in block i.

2) Upward Transfer: In the upward transfer, the challenge lies in properly initializing additional untrained layers to ensure seamless integration with the pretrained weights. We explore two different approaches: (1) Random initialization and (2) Parameter remapping. While the random initialization initializes parameters that are not covered by the pretrained architecture randomly, the parameter remapping applies established techniques that can transfer knowledge from small to big networks. Specifically, it does:

- **Kernel Expansion:** Smaller kernels are placed at the center of larger kernels, with the remaining weights initialized to zero (Fig. 4a).
- **Width Expansion:** Additional channels are initialized with zero as introduced in previous work. There, this zero-initialization has been shown to integrate well with existing weights (Fig. 4b).
- **Depth Expansion:** Network depth is expanded by duplicating the last pretrained block. The weights from the last block in the pretrained network are copied into additional subsequent blocks in the larger network (Fig. 4c).

While these heuristics for initializing untrained weights have proven effective in FNA, as an ablation study, we also explore the effect of randomly initializing untrained weights. We compare the performance of these two settings to test the adaptability of untrained weights.

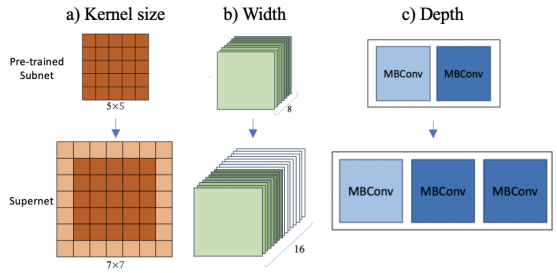


Fig. 4: Upward knowledge transfer across three search dimensions. For a) kernel size, the pretrained weights are zero padded. For b) width, additional channels are filled with zeros. For c) depth dimension, the last pretrained layer is repeated for all additional layers.

IV. Evaluation

A. Experimental Setup

We evaluate our SuperFast approach on ElasticViT [9] and NASViT [5] state-of-the-art search spaces on the ImageNet [6] dataset. For our experiments we build on the original code, training procedure and parameters as published on Github[1,2] at time of writing. Different to the original setup we reduce the number of GPUs used from 16 to 8 Nvidia V100 for ElasticViT and from 16 to 4 Nvidia A100 for NASViT for the execution of the experiments.

For pretraining specific submodels of the supernet, we follow the training parameters of [2] and stop the training after 50 epochs. All timing values are obtained by training a single epoch without validation on a single Nvidia A100 80GB GPU together with an AMD EPYC 7742 CPU.

B. Do Large Networks make for Better Initializers?

Past approaches have started their supernet training by warming up their supernet [1] or even relied on fully trained supernets [8]. Different to them we argue that using the biggest network as initialization for the supernet training is suboptimal as (1) it takes more time to train and (2) downscaling its features to smaller networks is difficult. Instead, in SuperFast we pretrain a medium sized architecture and use intelligent upscaling to preinitialize the search space more *efficiently* and *effectively*.

To verify this claim, we start by assessing the effectiveness of our intelligent upscaling method. For that, we initialize an ElasticViT search space with fully random weights (random-init), partially reloaded weights (direct-init) and our SuperFast intelligently initialized weights (SuperFast-init). We then train the supernet for one epoch to allow it to adjust itself to the loaded weights. The weights used for the initial knowledge are from a ElasticViT-M model, pretrained for 50 epochs. Table II shows the resulting average, minimum and maximum performance of the same 100 randomly sampled networks after 1 epoch of supernet training with the different initializations.

[1]https://github.com/facebookresearch/NASViT
[2]https://github.com/microsoft/Moonlit/tree/main/ElasticViT

Initialization	Top-1 (mean)	Top-1 (min)	Top-1 (max)
random-init	0.11	0.04	0.21
direct-init	7.68	**0.27**	53.55
SuperFast-init	**14.97**	0.24	**60.45**
supernet-init [8]	3.40	0.15	14.86

TABLE II: Performance of the same 100 random networks after 1 epoch of supernet training started from random-init, direct-init and SuperFast-init. The latter two are using knowledge from ElasticViT-M pretrained for 50 epochs. supernet-init uses a very large network for warm up, but falls short of ours.

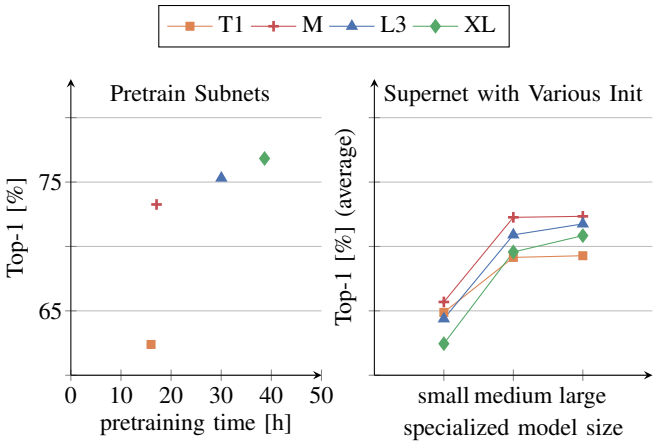


Fig. 5: Accuracy vs. training time trade-off when pretraining the T1, M, L3 and XL models of the ElasticViT search space for 50 epochs (left). Impact of the initialization on the average supernet performance for small, middle and large specialized networks after 10 epochs of supernet training (right). Very large networks take longer to pretrain and result in worse supernet performance, significantly reducing efficiency.

As we can see, random initialization barely moves beyond random predictions within the first epoch. Differently, the direct-init using the weights of the ElasticViT-M without any knowledge upscaling improves the performance significantly, resulting in a 7.57 p.p. increase of average top-1 and 53.34 p.p. increase of the best performing network over the fully random initialization. Our SuperFast-init improves beyond that: It surpasses the direct-init by 6.9 p.p. on the best performing network and by 7.29 p.p. on average, manifesting substantially better performance with our initialization. Finally, we compare with the initialization of the supernet of prior works [1], [8]. Specifically, we pretrain a ElasticViT-XL model, positioned on the top-end of the search space and use it to initialize the supernet. As can be seen in the table (supernet-init), using a very large network for the supernet initialization performs worse than just using the direct-init with ElasticViT-M, even without our upscaling.

Building on this finding, we move to investigate the impact of pretrained model size on the supernet training performance more carefully. For that, we pretrain architectures of four different sizes in different areas of the ElasticViT search

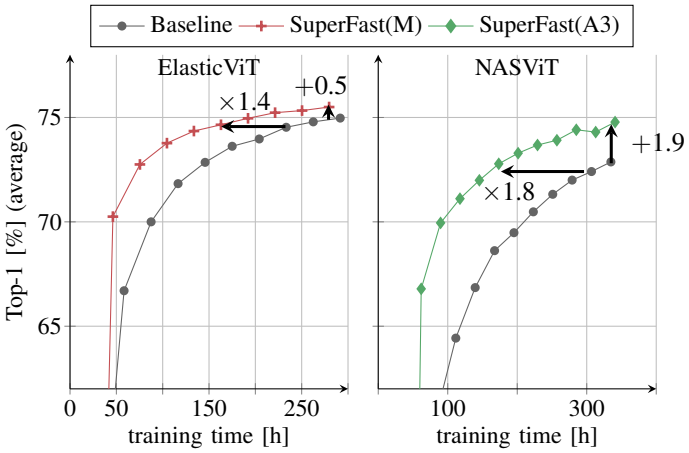


Fig. 6: Validation performance of the same 100 random architectures during the training of ElasticViT and NASViT supernets, with and without our SuperFast-init. In spite of the initial delay due to the pretraining, SuperFast achieves better accuracy vs training time trade-offs.

space. Specifically, we choose three predefined architectures ElasticViT-{T1, M, L3} at 62, 415, and 806 MFLOPs respectively. To properly represent the upper end of the search space, we again add the XL model, which has 1800 MFLOPs. Fig. 5 (left) shows the time cost and accuracy of all four models after 50 epochs of pretraining. As expected, both the accuracy and the training time of bigger models is higher. T1 is the only exception to this rule, as it does not saturate the compute of the GPU at the reference batch size. Notably, the M model achieves 95% of the accuracy of the XL model in less than half of the time, achieving a better accuracy vs. time trade-off.

Moving to the performance of the supernet using the different subnets as initialization, we find that the choice of subnet impacts the performance significantly. After 10 epochs of supernet training all supernet initializations result in relatively high performance already. In particular, we investigate the average top-1 accuracy of 100 random subnets that are grouped by their size into qualitative small, medium and large categories (Fig. 5, right). As can be seen there, the longest pretrained XL subnet with the highest accuracy does not result in the best performance. On the contrary, it falls behind the L3 and M subnets in all size categories, performing the worst of all of them for small subnets. The T1 initialized supernet is competitive to the M and L3 initialized supernet for small networks, but given its complexity is $\approx 20\times$ smaller than the largest sampled network, its ability of upscaling hits a limit. The medium sized M and L3 initialized supernets perform the best in most cases, striking a good balance between pretraining effort and upscaling ability. The fact that M initialized supernet outperforms the L3 initialized supernet which have similar pretraining performance, shows the remarkable efficiency of our approach and reinforces our hypothesis that scaling down features is generally harder than scaling up features.

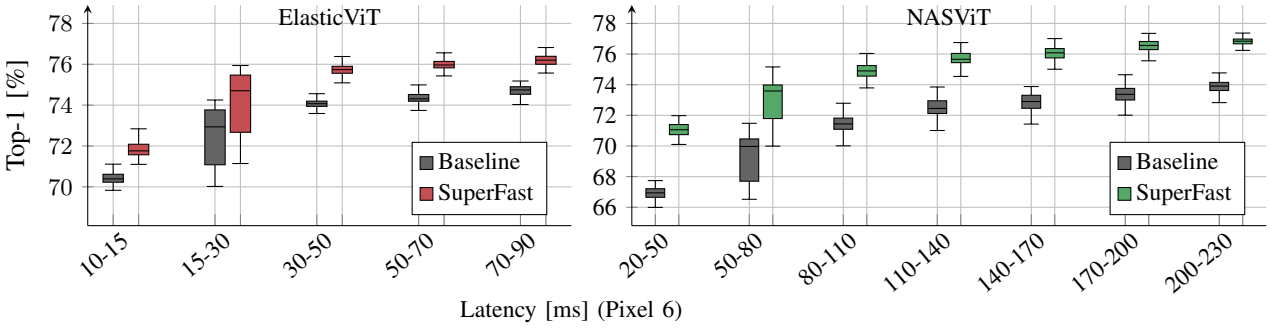


Fig. 7: Subnets sampled from ElasticViT and NASViT supernets, which were trained with- and without using our SuperFast procedure. As can be seen, our SuperFast procedure results in substantially higher accuracy for both search spaces on a wide range of latency budgets on Pixel 6.

C. Faster Supernet Training with SuperFast

After quantifying the advantages of using medium sized architectures with knowledge upscaling for initializing the supernet, we want to investigate the performance of our SuperFast approach on the ElasticViT and the NASViT search space. We start by investigating the time reduction and average accuracy gains achievable with our initialization. To evaluate both properties we train both supernets with the parameters in their public repositories for 1/6 of the total epochs. Afterwards, we repeat the same experiment with our SuperFast-init based on our pretrained ElasticViT-M subnet for the ElasticViT and a pretrained NASViT-A3 subnet for the NASViT supernet. NASViT-A3 was chosen as its position in the search space of NASViT is similar to the position of the M variant in the ElasticViT search space. Fig. 6 shows the accuracy over the time of training with and without our SuperFast-init for both search spaces. The plot accounts for the time it took to pretrain the subnets in both cases, adding a 33.91h and 17.1h offset for the SuperFast NASViT and ElasticViT training curves, respectively. In spite of the initial offsets, both SuperFast-init models catch up and overtake the baseline within the first 5 epochs after less than 50h and 60h for ElasticViT and NASViT respectively. During the course of the training, we observe substantial speedups as a result of our SuperFast-init. For ElasticViT we achieve the same accuracy $1.4\times$ earlier then the baseline. For NASViT the speedup is even greater, achieving a $1.8\times$ time reduction to reach the same accuracy. When using the same time budget for both, SuperFast improves the accuracy over the ElasticViT and NASViT baselines. After 250h and 350h the gain is 0.5 p.p. and 1.9 p.p., respectively with both seeing even higher gains when the time budget is shortened.

To further investigate if SuperFast supernet training can generate higher performing subnets across the entire search space beyond average improvements, we investigate subnets for multiple different latency budgets on the Google Pixel 6. We evaluate the baseline checkpoints of ElasticViT and NASViT after 146 h (50 epochs) and 279 h (50 epochs) of training and compare them to our checkpoints at 134 h (pretraining + 40 epochs) and 257 h (pretraining + 45 epochs) of training. We follow [9], to obtain accurate latency values for all subnets and divide the search space into 5 latency buckets for ElasticViT and 7 latency buckets for NASViT. For each bucket we sample 100 random networks and evaluate their accuracy. The results with and without SuperFast-init for both search spaces are shown in Fig. 7.

When evaluating our SuperFast-init results in Fig. 7, we see a clear improvement with SuperFast over both the NASViT and ElasticViT baselines for all latency budgets. In all but two cases, even the lowest quantile is above the highest quantile of the baseline, with the maximum distance between them being 2.3 p.p. When looking at the median values, the picture becomes even clearer with a minimum improvement of 1.4 p.p. and 2.9 p.p. and a maximum improvement of 1.8 p.p. and 4.0 p.p. for ElasticViT and NASViT, respectively. These performance improvements for all latency budgets demonstrate that SuperFast-init benefits *all* subnets in the search space. Looking beyond our own contribution, it is clear that the ElasticViT architecture design is very efficient, achieving *significantly* better accuracy latency trade-offs than its NASViT counterpart, therefore making it a much better choice for hardware deployment.

V. Conclusion

In this work, we introduced SuperFast, a plug-and-play procedure for supernets that accelerates the expensive supernet training, reducing its prohibitive cost. Different to previous works we do not warm up the largest network of the search space but find that scaling up features from small networks is easier than scaling down features from large ones. Consequently, for our approach, we made the choice to pretrain a medium sized model which is both more effective and more efficient. We demonstrated that our approach significantly improves supernet training time and accuracy on SOTA supernet search spaces, making it feasible to train them with limited resources. Future work may explore the usage of our method on other vision tasks, especially dense vision tasks, potentially demonstrating an even broader applicability of our method.

REFERENCES

[1] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020.

[2] Han Cai, Junyan Li, Muyan Hu, Chuang Gan, and Song Han. Efficientvit: Lightweight multi-scale attention for high-resolution dense prediction. In *ICCV*, pages 17302–17313, 2023.

[3] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[4] Jiemin Fang, Yuzhu Sun, Kangjian Peng, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Fast neural network adaptation via parameter remapping and architecture search. *arXiv preprint arXiv:2001.02525*, 2020.

[5] Chengyue Gong and Dilin Wang. Nasvit: Neural architecture search for efficient vision transformers with gradient conflict-aware supernet training. *ICLR*, 2022.

[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015.

[7] Manas Sahni, Shreya Varshini, Alind Khare, and Alexey Tumanov. Compofa: Compound once-for-all networks for faster multi-platform deployment. *arXiv preprint arXiv:2104.12642*, 2021.

[8] Sharath Nittur Sridhar, Souvik Kundu, Sairam Sundaresan, Maciej Szankin, and Anthony Sarah. Instatune: Instantaneous neural architecture search during fine-tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1523–1527, 2023.

[9] Chen Tang, Li Lyna Zhang, Huiqiang Jiang, Jiahang Xu, Ting Cao, Quanlu Zhang, Yuqing Yang, Zhi Wang, and Mao Yang. Elasticvit: Conflict-aware supernet training for deploying fast vision transformer on diverse mobile devices. In *ICCV*, pages 5829–5840, 2023.

[10] Dilin Wang, Chengyue Gong, Meng Li, Qiang Liu, and Vikas Chandra. Alphanet: Improved training of supernets with alpha-divergence. In *International Conference on Machine Learning*, pages 10760–10771. PMLR, 2021.

[11] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 702–717. Springer, 2020.

Hypnos: Memory Efficient Homomorphic Processing Unit

Haoxuan Wang*†§, Yinghao Yang*†, Hang Lu*†§✉, Xiaowei Li*†§
*SKLP, Institute of Computing Technology, Chinese Academy of Sciences,
†University of Chinese Academy of Sciences,
§Zhongguancun Laboratory, Beijing, China
{wanghaoxuan23p, luhang, lxw}@ict.ac.cn, yinghao.y@foxmail.com

***Abstract*—Fully Homomorphic Encryption (FHE) introduces a novel paradigm in privacy-preserving computation, extending its applicability to various scenarios. However, Operating on encrypted data imposes significant computational challenges, particularly elevating data transmission and memory access demands. Consequently, developing an efficient system storage architecture becomes vital for FHE-specific architectures. Traditional FHE accelerators use a Host+ACC topology, often focusing on enhancing computational performance and efficient using of on-chip caches, with the assumption that very large volumes of encrypted data are already present in the accelerator's memory while neglecting the inefficiencies of the unavoidable PCIe bus. In this paper, we propose Hypnos—a memory-efficient homomorphic encryption processing unit. In Hypnos, we abstract operators from FHE schemes into commands suitable for memory-efficient processing units and combine them with a homomorphic encryption paged memory management system designed for memory access, significantly reducing the memory access and execution time of homomorphic encryption applications. We implement Hypnos on the QianKun FPGA Card and highlight the following results: (1) outperforms SOTA ASIC and FPGA solutions by 2.58× and 4.43×; (2) the communication overhead is reduced by 3.78× compared to traditional architectures; (3) up to 27.6× and 19.06× energy efficiency improvement compared to ASIC-based CraterLake and FPGA-based Poseidon for ResNet-20 respectively.**

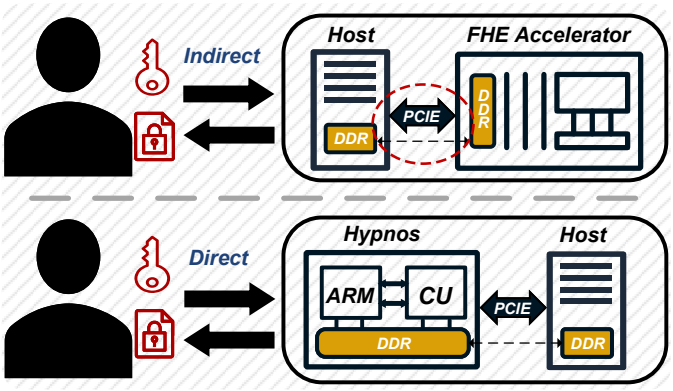


Fig. 1. Comparison with Traditional FHE Accelerators: In Hypnos, data processing is conducted directly by the CPU within the device, eliminating the dependency on the host CPU and bypassing the congested system PCIe bus.

I. Introduction

With the rapid development of the internet and cloud computing, privacy protection and security have become major challenges in information processing. Against this backdrop, the importance of data privacy computation technology has become increasingly prominent. Fully Homomorphic Encryption (FHE) is an important privacy computation technology that is crucial in achieving secure computations [1], [2].

The primary advantage of FHE is its ability to perform unlimited addition and multiplication operations without decrypting the data. This allows complex computations to be executed while the data remains encrypted. However, this characteristic of FHE results in an explosive expansion in data volume and computational complexity, ranging from 10,000 to 100,000 times. This places considerable strain on memory and computational systems, making it impractical for general-purpose computing hardware to handle FHE computations effectively. Therefore, it is essential to develop specialized architectures to enhance the efficiency of FHE computations.

In recent years, several FHE-specific accelerator architectures have been for FHE acceleration based on FPGA [3], [4] and ASIC [5]–[9]. The advantage of using FPGAs lies in their lower cost and programmable nature, which allows for flexible iterations and upgrades of hardware designs to achieve rapid deployment in response to the evolving FHE algorithms. In contrast, ASICs deliver higher performance and lower power consumption, allocating more substantial hardware resources to address the bottlenecks in FHE, such as employing up to 180MB-512MB of on-chip scratchpad to mitigate the overhead caused by data transfers between off-chip memory and computational units (CUs) [6]–[8]. However, these ASICs are currently in the simulation phase, making it difficult to rapidly deploy them for real-world use.

Prior works have typically assumed that substantial volumes of ciphertext and keys are always stored within the DDR/HBM of the accelerator. However, this assumption is overly idealistic, as handling intricate FHE applications often exceeds the capacity of on-chip memory. This indicates that communication between the host and the accelerator via the PCIe bus will be frequent. As depicted in the upper section of Fig.1, communication between the user and the accelerator is mediated by host memory, necessitating data to traverse the PCIe bus after initial storage in host memory. This bottleneck can potentially undermine the performance gains reported in earlier studies.

Current designs of FHE accelerator architectures emphasize enhancing computational efficiency and optimizing the utilization of on-chip memory [10]. They often neglect the utilization efficiency of off-chip memory. In fact, in FHE schemes based on Residue Number Systems (RNS), the sizes of ciphertexts and keys vary dynamically during computation. Allocating and managing memory with a fixed size will lead to a large amount of redundant information being stored in memory, thereby reducing memory utilization efficiency. It will exacerbate the overhead of PCIe communication.

Haoxuan Wang and **Yinghao Yang** are co-first authors. **Hang Lu** is the corresponding author. Bo Wang <wangbo@phytium.com.cn>, Changshan Su <suchangshan1101@phytium.com.cn>, Xiaoyu Li <lixiaoyu2261@phytium.com.cn>, Gen Li <ligen@phytium.com.cn>, Yuxing Tang <tangyuxing@phytium.com.cn> (all with the Phytium Information Technology Co., LTD) contribute significantly to this paper in terms of the manuscript submission and rebuttal, and should all be regarded as co-authors.

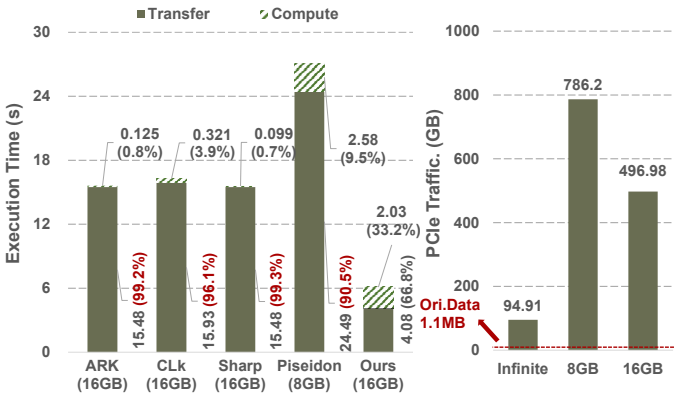


Fig. 2. Execution time breakdown of ResNet-20. The data size under FHE encryption is severely inflated compared to plaintext (1.1MB marked in red), introducing significant PCIe communication overheads. PCIe transfer overhead is high for existing accelerator architectures, over 96% for ASIC accelerators and up to 90% for FPGA-based Poseidon.

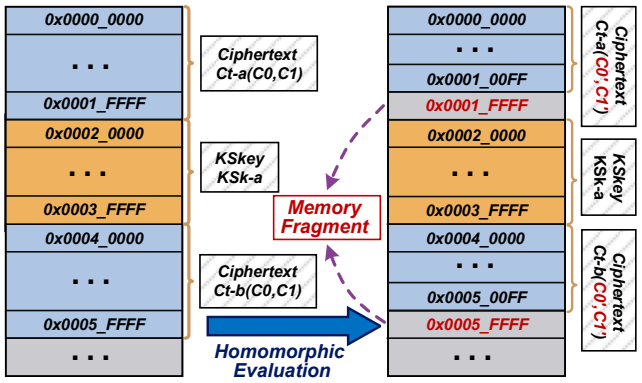


Fig. 3. Memory fragmentation during homomorphic computation. Changes in the number of RNS components of ciphertext lead to memory fragmentation.

In this paper, we propose Hypnos, a memory-efficient homomorphic processing unit that changes the way computational units access encrypted data and significantly reduces the amount of ciphertext and key transmission. For RNS-based FHE applications where the data volume exceeds the off-chip memory, we propose a homomorphic encryption page memory management system, which reduces memory fragmentation and the volume of data communicated with the host.

In summary, our contributions are as follows:

- We propose a practical, memory-efficient homomorphic processing unit named Hypnos. It aims to mitigate the performance overhead caused by inefficient PCIe communication and improve off-chip memory utilization.
- We propose a novel memory management system based on "HE page" to manage memory at the RNS component level, significantly increase memory utilization, and further reduce the data transfer volumes during the execution of FHE applications.
- We implement a prototype of Hypnos on the commercial Qiankun, and evaluate its performance in practical FHE applications. We highlight the following results: (1) Hypnos outperforms SOTA ASIC and FPGA solutions by 2.58× and 4.43×, respectively; (2) communication overhead is reduced by 3.78× compared to traditional accelerator architectures; (3)In terms of energy efficiency, Hypnos also surpasses existing ASIC and FPGA accelerators by a factor of 27.6× and 19.06×, respectively.

II. Background & Motivation

A. Platform Limitations for FHE Accelerators

Due to the on-board memory limitation problems, accelerators must frequently transmit large volumes of ciphertext and keys when processing FHE applications. This extensive data movement between the host DDR and the accelerator's local memory can severely impact performance. Prior works have often underestimated this issue. In our evaluation of the ResNet-20 application based on the SEAL library, as depicted in the left half of Fig. 2, despite the superior computational efficiency of ASIC accelerators, the overhead associated with PCIe transfers can account for up to 96% of the total execution time. Similarly, FPGA accelerators confront analogous issues attributable to their limited onboard memory. The right side of Fig. 2 shows the increase in PCIe communication overhead under different onboard memory sizes, where the "infinite" item represents the communication scenario when all data is loaded into the onboard memory at once. The results show that, under memory constraints, PCIe communication volume can surge dramatically. Therefore, reducing PCIe communication overhead is a critical consideration for FHE accelerators. Otherwise, continuously optimizing the performance of the accelerators alone will not eliminate the PCIe communication bottleneck within the entire acceleration system.

To address this bottleneck, we propose integrating a CPU onto the accelerator to manage the direct transmission of ciphertexts and keys, thereby reducing the overhead associated with PCIe communications. Hypnos exhibiting similar computational performance compared to the FPGA-based accelerator Poseidon achieves superior overall performance due to its reduced PCIe overhead. Even when benchmarked against more computationally powerful ASICs, Hypnos can elevate system performance through optimized PCIe communication.

B. Neglected Memory Fragmentation

In RNS-based FHE schemes, ciphertexts and keys are represented as polynomials. As shown in Fig. 3, during computations, the number of polynomials changes, leading to variations in ciphertext size and the number of RNS components required for keys. Fixed-size memory management results in memory fragmentation and decreased utilization, particularly evident in applications like ResNet-20 and LR-Train using the CKKS scheme, where memory efficiency is notably reduced, increasing the frequency of data replacement and PCIe communication overhead. However, the BFV scheme, with its constant ciphertext and key sizes, is not affected by this issue.

Fortunately, the size of the RNS components in RNS-based FHE schemes is predetermined, meaning that all ciphertexts and keys can be considered composed of multiple fixed-size data segments. This provides an opportunity for optimizing memory utilization efficiency.

III. Hypons Architecture

A. Architecture Overview

As shown in Fig.4, the Hypnos architecture comprises three distinct regions: the Processing System (PS), Programmable Logic (PL), and Network-On-Chip (NOC).

The PS region is equipped with a CPU that runs an operating system, responsible for initializing the state of the compute units and executing FHE computation applications. This includes handling FHE operations, data transfers, and monitoring the status of the compute units.

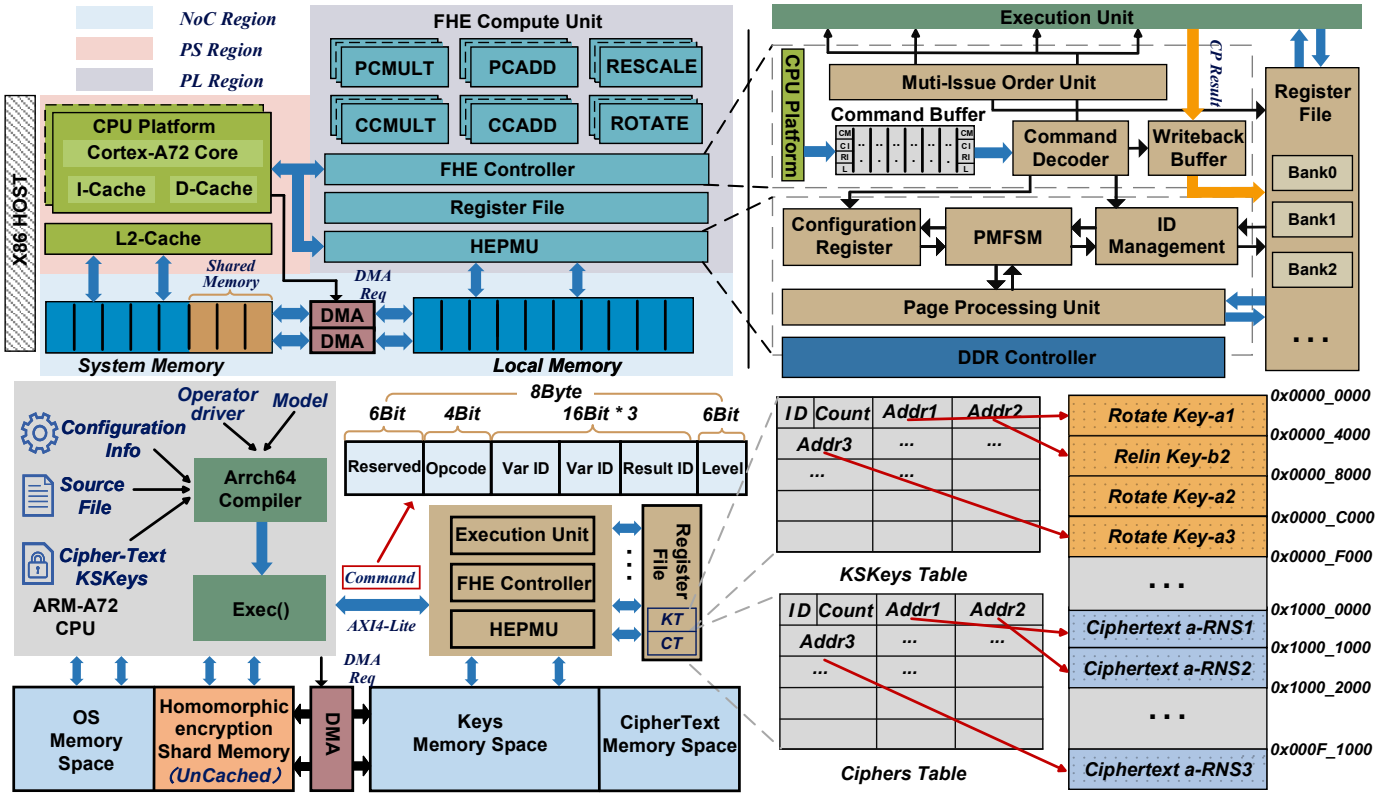


Fig. 4. Hypnos overall architecture. The heterogeneous ARM+FPGA computing architecture features a Cortex-A72 CPU, connected through a NoC to various components including the FHE CUs, HEPMU, and memory subsystems. This setup highlights the multi-layered interaction between processing units, local and system memories, DMA channels, and specialized encryption hardware, illustrating a memory-efficient system for FHE acceleration.

The PL region offers reconfigurable hardware resources serving as compute unit(CU), composed of six fundamental FHE operators that support computations for various FHE schemes. The CU includes an FHE controller, register file, and Homomorphic Encryption Page Management Unit(HEPMU), responsible for command execution, caching, and memory management, respectively.

The NOC region acts as the interconnection network, utilizing the on-chip buses of the AMD Xilinx Versal ACAP series to facilitate high-speed communication between components and support point-to-point data transfers.

B. FHE Execution&Controller Unit

Hypnos adopts an abstracted methodology utilizing operational-level operators, as opposed to basic operators (such as NTT, MM, MA, etc.), providing 64 parallel processing units. Within these operational-level constructs, basic operators function in a cascaded arrangement, with each unit possessing its own cache, thereby mitigating accesses to main memory and enhancing the efficiency of request processing alongside the overall system performance. Internally, the Barrett Reduction algorithm [11] is employed for modular multiplication (MM), while the Number Theoretic Transform (NTT) utilizes a radix-8 design and incorporates optimizations derived from Poseidon [4].

The FHE controller is responsible for scheduling, whereas the command decoder assigns tasks to the functional units. The role of the Multi-Issue Order Unit is to disseminate processed command information to the CUs. Computed results are subsequently stored in registers via a write-back buffer. The HEPMU manages memory operations and returns results either to the CPU or for further computations.

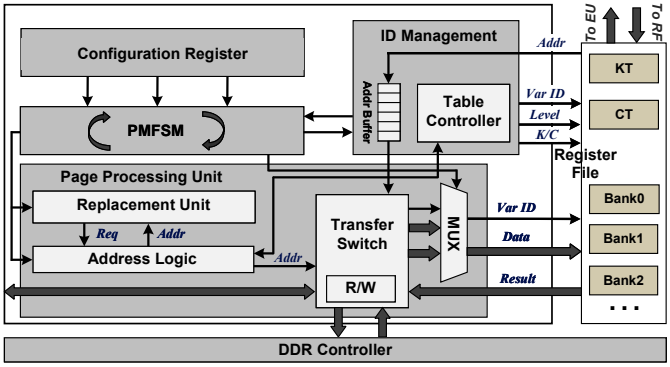


Fig. 5. Micro-architecture of Homomorphic Encryption Page Management Unit (HEPMU).

C. HEPMU

The cornerstone of Hypnos is the HEPMU, which implements paged memory management at the granularity of RNS components. The microarchitecture of the HEPMU is illustrated in Fig.5. The detailed workflow of the HEPMU will be elaborated upon in Sec.IV; herein, we focus solely on the hardware architecture design.

Configuration Register provides scalable memory management granularity and table entries of varying sizes based on different applications and security parameters. Configuration details are written

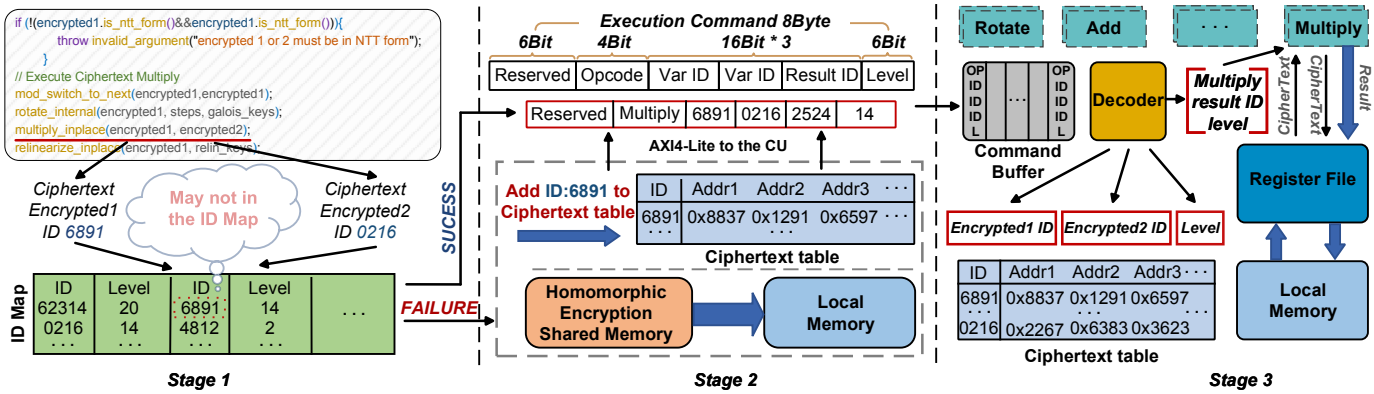


Fig. 6. This diagram illustrates the multi-stage execution process of a command within Hypnos, providing a detailed account of the entire workflow. Each stage is managed and invoked via specific operation IDs, addresses, and level parameters, ensuring the accurate execution of the encrypted command.

via the AXI4-Lite bus and are utilized during the execution of FHE applications. Additionally, the CPU can read this configuration information through the same bus to determine the status of Hypnos.

Page Management Finite State Machine (PMFSM) acts as the central control logic, switching between idle, write, eviction, and read states according to the sequence of commands, coordinating task scheduling within the HEPMU.

ID Management performs ID-to-address translation, querying addresses from the Key Table (KT) and Ciphertext Table (CT) based on the type of ciphertext variable, and writing addresses to the buffer. Notably, the CT and KT are stored in a register file to accommodate dynamic changes in table sizes due to varying parameter configurations. For example, when the modulo chain length $L = 32$ and the memory size is 16GB, the sizes of CT and KT are 69KB and 17.25KB, respectively.

Page Processing Unit (PPU) executes operations related to the RNS page table, retrieving RNS components based on address information and placing them in the register file for computation. When page replacement is required, it writes page information back to the KT or CT.

It is noteworthy that, as shown in the lower-left portion of Fig.4, a portion of the system memory is designated as shared memory, specifically reserved for storing ciphertexts and keys, and configured as uncacheable. This design choice is because, during homomorphic computations, the CPU processes address information rather than the ciphertexts and keys themselves. Moreover, the uncacheable attribute of the shared memory eliminates the computational overhead associated with cache coherence checks and cache flushing. Given that different applications have varying requirements for ciphertext and key memory spaces, the sizes of these spaces are dynamically specified by the FHE program running on the CPU. For instance, in CNN applications that require a large number of keys, a larger key storage space can be allocated to mitigate the need for frequent data exchanges.

Regarding the design of page table replacement strategies in the HEPMU, we evaluated several strategies for ResNet-20 on Hypnos. The performance of ResNet-20 indicated that, both in terms of execution time and the number of replacements, the Least Recently Used (LRU) strategy demonstrated performance closest to the optimal (OPT). Consequently, we adopted LRU as the page table replacement strategy in the HEPMU.

IV. HE Paged Management System

In prior accelerator designs, despite the adoption of RNS decomposition, ciphertexts and keys were still transmitted and stored as whole units. Only during the execution phase were the necessary RNS components used for computation, leading to unused RNS components inefficiently occupying memory space. To address this, we propose a memory-efficient homomorphic paging memory management system designed to manage ciphertexts and keys in memory with precision, thereby optimizing memory usage and enhancing performance.

This system operates at the level of RNS components and achieves software-hardware collaboration. The CPU is responsible for executing the top-level homomorphic encryption application code, while the CU manages the physical addresses of homomorphic variables and executes FHE operators. The close collaboration between the CPU and CU is key to achieving efficient computation. Fig. 6 illustrates the execution flow of homomorphic encryption application computations, including the details of memory management.

CPU Side: FHE applications execute on the CPU after specialized compilation. As shown in Stage ❶ of Fig. 6, when the program requires acceleration via FHE operators, it first queries the *ID Map* maintained on the software end to verify whether the required variables exist in the *CU* memory. In Stage ❷, if all variables are present, the program sends the command data to the CU and waits for a result signal. In the lower part of Stage ❷, if any variables are missing, they are sent to the *CU*, and the *PPU* updates the variable IDs and their levels in the *ID Map* and the *CT* or *KT*. During this process, the *CPU* only accesses the *IDs* and *Level* of the variables and awaits an interrupt signal indicating the completion of the computation.

Compute Unit Side: Address management, data replacement, and write-back within the memory system are all performed on the *CU* side. Upon receiving a command, the compute unit verifies that all ciphertext variables are present in memory. It then retrieves and decodes the command content from the *Command Buffer*, using the *IDs* to locate the physical addresses of the variables in the *CT* and *KT*. The *Level* indicates the required depth of multiplication. The *HEPMU* transfers the variables to the register file, and the decoder inputs the type of computation and auxiliary information to the operator. Once the computation is complete, the *result ID* and the computed result are written back to the *register file*, as shown in Stage ❸. The *CU* notifies the *CPU* of the result and its physical address via an interrupt. The *CPU* then fetches the result to complete the command execution cycle.

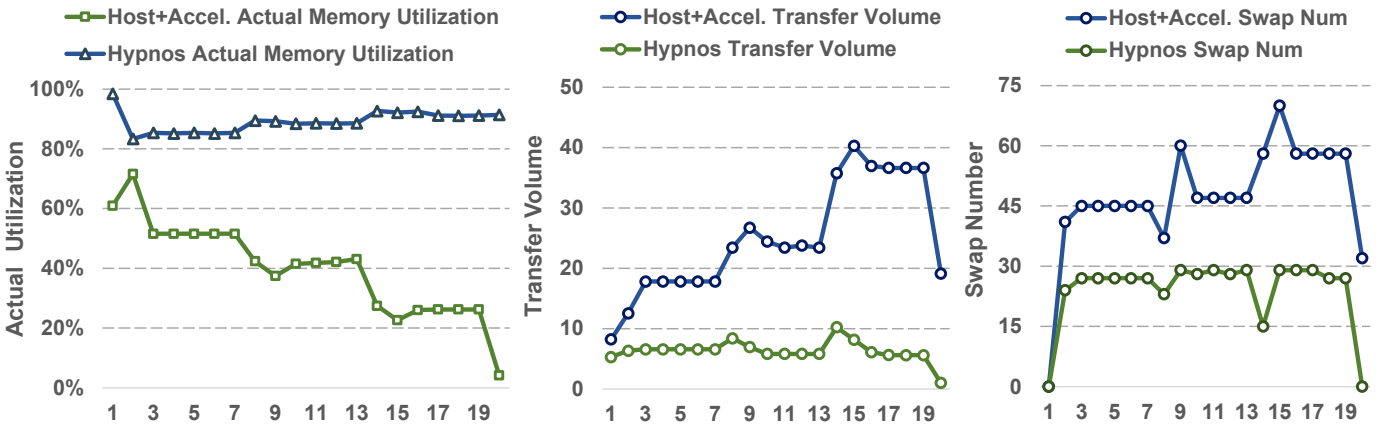


Fig. 7. How (a) memory utilization, (b) transfer volume, (c) swap number change at each layer of ResNet-20 under encrypted execution when comparing Hypnos to traditional architectures, with both systems featuring a uniform memory size of 16GB and differing in granularity: the traditional architecture managing memory at the granularity of entire ciphertexts, while Hypnos operates at the granularity of a single RNS component.

TABLE I
PERFORMANCE COMPARISON OF FOUR BENCHMARK APPLICATIONS WITH THE BASELINE FHE ACCELERATOR, MEASURED IN TWO TIME UNITS: SECONDS (S) AND MILLISECONDS (MS)

Works	ResNet-20 (in s)	LR-Train (in s)	PSI (in ms)	PIR (in ms)
(ASIC) ARK	15.61	0.34	422.73	306.10
(ASIC) Craterlake	15.80	0.25	408.45	303.97
(ASIC) Sharp	15.58	0.18	417.73	305.23
(FPGA) Poseidon	27.07	2.32	689.79	389.77
(FPGA) FAB	22.68	3.22	1168.12	472.30
(FPGA)Hypnos	**6.11**	**1.89**	**278.27**	**86.72**

V. EVALUATION

A. Experimental Setup

Platform. Hypnos is implemented on the Qiankun FPGA Card from Ant Inc., featuring a dual-core ARM A72 processor, over 1.7 million LUTs, and more than 7000 DSPs. The specific FPGA model is the xcvp1502-vsva2785-2MHP-e-S.

Baseline. In our experiments, Hypnos was compared with state-of-the-art ASIC-based FHE accelerators, including ARK [6], Craterlake [5], and SHARP [8], and FPGA-based accelerators such as FAB [3] and Poseidon [4]. Some accelerators only implemented portions of the Benchmark. For clarity, we evaluated these architectures based on the complexity of each Benchmark.

Benchmark: We use the following 4 benchmarks (as shown in TableI) for evaluation:

(1) ResNet-20: Implemented and trained with the CIFAR-10 dataset, featuring 19 convolutional layers and one fully-connected layer, with ReLU activation functions [12].

(2) Logistic Regression (LR): Based on the HELR algorithm using the CKKS scheme [1], incorporating Bootstrapping with a multiplication depth of $L = 38$ evaluated over 10 iterations supported by two Bootstrapping operations.

(3) Private Set Intersection (PSI): Evaluated using the SEAL APSI open-source library for computing the intersection of two asymmetric sets, with the sender's set containing 2^{28} elements and the receiver's set 2^{10} elements [13].

(4) Private Information Retrieval (PIR): Benchmark evaluates the SEAL PIR open-source library for encrypted queries on the server side, involving a query for a single item from a dataset of 2^{20} entries [14].

B. Full-system Performance

The design goal of Hypnos is to provide an architecture capable of running complete FHE applications. We demonstrate the full-system performance of Hypnos in comparison to previous accelerators. As shown in Table I, for data-intensive applications such as ResNet-20, PSI, and PIR, the primary bottleneck is the overhead of memory access. Hypnos can leverage its strengths, achieving up to a 2.58× improvement in performance on ASIC-based accelerators and up to a 4.4× improvement on FPGA-based accelerators. LR-Train, on the other hand, is a HE application that is computationally intensive with minimal memory access; it does not involve large-scale databases or extensive rotating keys. In terms of computational performance, Hypnos achieves a 1.7× improvement over FPGA accelerators, though a performance gap remains compared to ASIC accelerators.

C. Energy Efficiency

We utilize Energy-Delay Product (EDP) as the metric for energy efficiency. TableII presents the energy efficiency performance of Hypnos compared with ASIC-based accelerators ARK and Crakelake, and the FPGA-based accelerator Poseidon across four benchmarks. In ResNet-20, PSI and PIR, Hypnos outperforms both ASIC and FPGA accelerators, achieving enhancements of 27.6×, and 19.06× respectively. This is primarily due to the substantial data transfers, which contribute to significant PCIe overhead. Hypnos' memory-efficient architecture excels in managing these demands. However, in the LR-Train applications, Hypnos does not match the performance of ASIC accelerators in terms of EDP, due to FPGA's limited resources, yet it still surpasses the FPGA accelerator Poseidon.

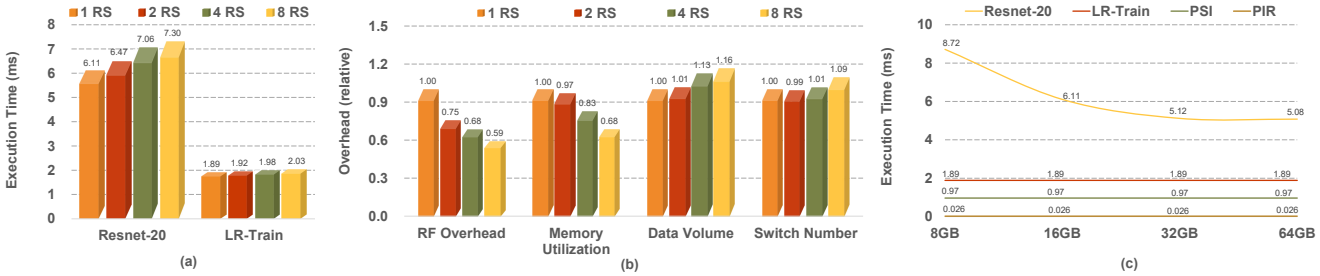


Fig. 8. Analysis of sensitivity. (a) and (b) explore the granularity of memory management within ResNet-20 and LR-Train. (c) examines the impact of memory size on the Hypnos for selected benchmarks.

TABLE II
EFFICIENCY ANALYSIS. WE USE ENERGY-DELAY PRODUCT (EDP) AS THE METRIC. LOWER IS BETTER.

Work	ResNet-20	LR-Train	PSI	PIR
ARK	68545.1	32.5	50267.4	26356.8
Craterlake	51675.5	12.9	34533.3	19125.8
Poseidon	87619.1	643.6	56892.1	18165.3
Hypnos	**4730.1**	**452.6**	**9810.7**	**952.8**

TABLE III
HARDWARE RESOURCE UTILIZATION

Resources	LUT(K)	FF(K)	BRAM	DSP
Used	860	1,115	2,511	4,990
Percentage	(50.02%)	(32.41%)	(65.36%)	(67.07%)

D. Memory

In prior work, both ASIC and FPGA-based FHE accelerators utilize a Host as the master end for HE applications, necessitating data transfers via the bandwidth-limited PCIe protocol. Consequently, we abstract this arrangement into a Host+Accelerator model. Fig.7 uses ResNet-20 as an example, with the CNN layers represented on the horizontal axis, to compare and analyze the advantages of the Hypnos architecture over traditional architectures through three metrics.

(1) Memory Utilization: Defined as the percentage of RNS components used for computation at each layer. For example, with L_{max}=30 and an encryption operation level of 10, the MU is 33.33%. Traditional accelerators exhibit low MU due to unused data occupying memory; MU can drop below 20% at the final layer. Hypnos does not reach 100% MU because some variables are frequently reused in higher-level computations, optimizing which would degrade performance.

(2) Transfer Volume: Refers to the data volume transferred via PCIe at each layer. As shown in Fig.7 (b), initially, Hypnos exhibits similar transfer volumes to traditional architectures since variables are fresh ciphertexts at higher levels. Mid-layer volumes are reduced by 73.55% due to Hypnos' homomorphic encryption page-based memory management. At the final layer, volumes decrease sharply as it is a fully connected layer requiring less data.

(3) Swap Number: Counts the number of memory variable swaps due to insufficient memory. The decrease in swap frequency can be attributed to the increase in MU. As shown in Fig.7 (c), Hypnos decreases the total number of swaps by more than 50% compared to traditional architectures.

E. FPGA Resource Utilization

Hypnos operates on an ARM+FPGA heterogeneous computing platform. TableIII delineates the resource utilization within the PL region. This includes the CU, controllers, register files, and the HEPMU. A significant portion of the DSP blocks are allocated to operator computations, whereas the DSP blocks within the HEPMU primarily support the hardware implementation of the LRU replacement algorithm and manage physical addresses. In the context of the HEPMU, BRAM is utilized for caching homomorphically encrypted variables. It is worth noting that the ARM processor, NoC, and DDR controller on the FPGA card are implemented as hard IP cores and thus do not consume any of the FPGA's programmable resources.

F. Sensitivity

Fig. 8 explores the (a) (b) impact of memory management granularity based on RNS components and (c) the size of the memory. Larger RNS granularity leads to increased memory fragmentation, reducing FHE storage system efficiency and overall performance. Using ResNet-20 as an example, increasing the granularity to L_{max} diminishes memory utilization, increasing data transfer volume and swap frequency to pre-optimization levels. For applications with large data volumes, increasing off-chip memory from 8GB to 16GB significantly reduces execution time, with minimal gains beyond 16GB. Smaller data volume applications are less sensitive to memory size, indicating that memory size allocation should be tailored to meet specific application requirements

VI. CONCLUSION

In this paper, we implemented the memory-efficient homomorphic processing unit Hypnos on the FPGA card. By integrating a CPU, the CU can directly access homomorphically encrypted data. Additionally, we introduce a homomorphic encryption paging memory management system that operates at the granularity of RNS components to manage memory, thereby reducing data movement during execution. Hypnos demonstrates remarkable efficiency in performing data-intensive computations. Compared to FHE accelerators based on ASICs, utilizing FPGAs as an implementation platform has more practical significance, enabling lower-cost updates and iterations of FHE algorithms. We hope our work will inspire new design concepts for domain-specific accelerators tailored for FHE applications.

VII. ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 62172387; in part by the CCF-Phytium Fund 2023.

REFERENCES

[1] K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic regression on homomorphic encrypted data at scale," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 9466–9471.

[2] H. Chabanne, R. Lescuyer, J. Milgram, C. Morel, and E. Prouff, "Recognition over encrypted faces," in *Mobile, Secure, and Programmable Networking: 4th International Conference, MSPN 2018, Paris, France, June 18-20, 2018, Revised Selected Papers 4*. Springer, 2019, pp. 174–191.

[3] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 882–895.

[4] Y. Yang, H. Zhang, S. Fan, H. Lu, M. Zhang, and X. Li, "Poseidon: Practical homomorphic encryption accelerator," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 870–881.

[5] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data." in *ISCA*, 2022, pp. 173–187.

[6] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.

[7] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "Bts: An accelerator for bootstrappable fully homomorphic encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 711–725.

[8] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[9] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 238–252.

[10] R. Agrawal, L. De Castro, C. Juvekar, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "Mad: Memory-aware design techniques for accelerating fully homomorphic encryption," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 685–697.

[11] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[12] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[13] "Microsoft APSI," https://github.com/microsoft/APSI, Dec. 2023, microsoft Research, Redmond, WA.

[14] "Microsoft SealPIR," https://github.com/microsoft/SealPIR, Jun. 2023, microsoft Research, Redmond, WA.

A PulseWidth-IN-PulseWidth-Out Universal Nonlinear Processing Element for Time-Domain In-Memory Computing Systems

Yihao Chen[1,3], Pengcheng Feng[1,3], Zhigang Li[1,3], Gang Chen[1,3], Rongxuan Shen[1,3], Huaxiang Lu[1,3*], Xiaoxin Xu[2,3]
[1]Institute of Semiconductors, Chinese Academy of Sciences; [2]Institute of Microelectronics, Chinese Academy of Sciences;
[3]University of Chinese Academy of Sciences.
*Corresponding author: luhx@semic.ac.cn

***Abstract*—Time-Domain In-Memory Computing (TD-IMC) has emerged as a promising analog computing architecture for edge AI applications. However, the lack of developed hardware operators, especially general nonlinear operators, necessitates frequent cross-domain data transmission in practical TD-IMC systems, significantly reducing energy efficiency. In this work, we propose a PulseWidth-IN-PulseWidth-OUT Universal Nonlinear Processing Element (PIPO-UNPE) to address the challenges of nonlinear processing in analog computing. By implementing an RRAM-based two-layer ReLU network, the PIPO-UNPE performs universal nonlinear operations entirely in the time domain. Algorithmically, we introduce Dynamic Loss-Responsive Subset Enhancement (DLRSE) to boost the performance of this low-cost network in function approximation tasks. From a hardware perspective, we design an RRAM-based pulse-driven programmable current source and a low-latency dispersion comparator-based voltage-to-time converter (VTC) to enhance both the energy efficiency and precision of the PIPO-UNPE. Hybrid simulations reveal that the PIPO-UNPE consumes 912 uW of power while delivering a throughput of 10M NOPS (Nonlinear Operations Per Second). Incorporating the PIPO-UNPE into the TD-IMC accelerator can increase energy efficiency by a factor of 9.5 to 25, keeping the accuracy loss below 0.1%.**

***Index Terms*—Time-Domain In-Memory Computing, Universal Nonlinear Processing Element, Neural Network Training, RRAM**

I. Introduction

Time-Domain In-Memory Computing (TD-IMC), a promising computing architecture based on pulse-width modulation has been widely applied in Edge AI applications [1]. In TD-IMC systems, the multiply-accumulate (MAC) operations of Deep Neural Networks (DNNs) are executed in the analog domain with high energy efficiency, and the MAC results are represented and transmitted in the time domain using pulse widths. Compared to other IMC architectures, TD-IMC not only benefits from the improved time resolution offered by advanced process nodes, but also enables large-scale integration thanks to its simple data routing that only requires logic circuits [2].

Recent TD-IMC systems have primarily concentrated on optimizing the MAC macro. However, nonlinear operation in DNNs, such as the computations of the activation functions (AFs) and their derivatives (DAFs) during DNN inference and on-chip training respectively [3], have still relied on digital processing elements (DPEs) for their complexity [4]–[7], necessitating the extensive use of power-hungry cross-domain modules like time-to-digital converters. To improve the energy efficiency of the TD-IMC, implementing an Analog Universal Nonlinear Processing Element (A-UNPE) for the direct nonlinear processing of pulse widths is essential. Notwithstanding, reported A-UNPEs [8]–[10], typically implemented by CMOS polynomial circuits, face challenges balancing high power consumption and limited support for various functions, stemming from the implementation of critical polynomial weights through digital-to-analog converters (DACs).

Instead of resorting to more complex polynomials, the ReLU neural network, which can be efficiently implemented in TD-IMC systems, has been proven to be an excellent universal function approximator [11]. An A-UNPE based on the ReLU neural network is expected to benefit from the inherent energy efficiency and scalability of the TD-IMC. Nonetheless, previous research has shown that simple ReLU networks struggle to achieve high approximation precision through naive backpropagation necessitating more complex network models and consequently increasing hardware costs [12].

In this work, the Dynamic Loss-Responsive Subset Enhancement (DLRSE) strategy is proposed to improve the approximation accuracy of the ReLU network. A PulseWidth-IN-PulseWidth-Out UNPE (PIPO-UNPE) based on the trained network is introduced. The key circuits of PIPO-UNPE include the MAC array constructed using RRAM-based Pulse-driven Programmable Current Sources (R-PPCS), as well as the Differential-IN-Differential-Out VTC (DIDO-VTC) and Single-IN-Differential-Out VTC (SIDO-VTC) implemented with low-latency dispersion comparators. Through hardware-software co-optimization, the PIPO-UNPE achieved an average precision of 9 bits when approximating multiple activation functions (ELU, GELU, Mish, Sigmoid, SiLU, and Tanh), while consuming 912 uW of power and delivering a throughput of 10M NOPS. The simulation model was tested on the state-of-the-art TD-CIM architecture TIMELY [13] to evaluate the energy improvement and inference accuracy. The simulation results show that PIPO-UNPE can improve energy efficiency by factors ranging from 9.5× to 25× while incurring

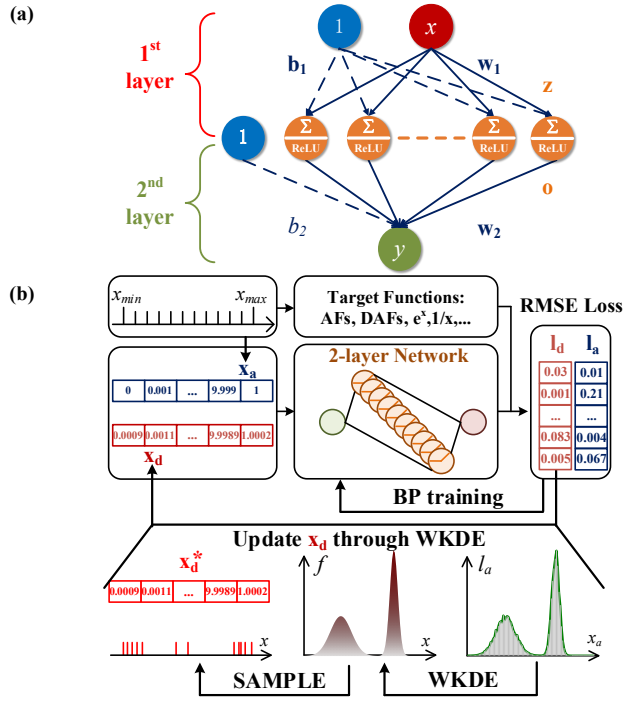


Fig. 1. (a) The proposed two-layer ReLU network for function approximation. (b) The completed training process using the DLRSE data augmentation strategy.

TABLE I
PERFORMANCE COMPARISON AND ABLATION STUDY RESULTS

Function	Domain	MAE ([12])	R_{MAE} (N=16)			R_{MAE} (N=32)		
			DLRSE	(w/o) $\mathbf{x_d}$	(w/o) $\mathbf{x_a}$	DLRSE	(w/o) $\mathbf{x_d}$	(w/o) $\mathbf{x_a}$
e^x	(-6,6)	0.096	1.4	2.8	3.3	0.6	0.9	1.6
$1/x$	(1,3)	0.0002	0.9	331.5	255.5	0.4	235.0	268.5
$1/\sqrt{x}$	(1,9)	0.0002	1.2	73.5	88.0	0.5	73.5	93.0
$1/(1+\sqrt{x})$	(1,9)	0.0001	0.9	48.2	55.6	0.3	45.3	53.1
Sigmoid	(-6,6)	0.0004	1.6	7.4	9.7	0.6	5.4	6.4
ELU	(-6,6)	0.0001	3.4	18.9	22.9	1.5	18.2	29.2
Tanh	(-6,6)	0.0006	1.8	5.1	7.2	0.6	3.0	3.7

[1] R_{MAE} is calculated as the ratio of the proposed model's MAE to the MAE of [12].
[2] The model in [12] contains $P = 5.793$k parameters, whereas the proposed model has only $P = 48$ parameters when $N = 16$ and $P = 97$ parameters when $N = 32$.

an accuracy loss of less than 0.1%.

II. PROPOSED MODEL AND TRAINING STRATEGY

In our work, an IMC-friendly neural network, specifically a two-layer ReLU neural network, is embedded in the PIPO-UNPE as its core computational mechanism. As illustrated in Fig. 1(a), the first layer in the proposed model consists of N hidden neurons that execute an affine transformation $\mathbf{w_1}x+\mathbf{b_1}$ on the scalar input x, mapping it to an intermediate vector $\mathbf{z}$. This vector is then transformed into a nonlinear vector $\mathbf{o}$ using the ReLU activation function. In the second layer, $\mathbf{o}$ undergoes a transformation $\mathbf{w_2}\mathbf{o}+b_2$, resulting in the approximated value y. Due to the limited effectiveness of naive training strategies in achieving high precision for function approximation [12], [14], we introduce the Dynamic Loss-Responsive Subset Enhancement (DLRSE) data augmentation strategy, as depicted in Fig. 1(b). DLRSE comprises a static anchor dataset $\mathbf{x_a}$ and a dynamically updated dataset $\mathbf{x_d}$. The $\mathbf{x_a}$ is obtained by uniformly selecting points within the fitting interval and remains unchanged during training, while $\mathbf{x_d}$ is updated every epoch based on the loss of $\mathbf{x_a}$ using Weighted Kernel Density Estimation (WKDE) to transform the loss into a probability density function, from which $\mathbf{x_d^*}$ is sampled. By exploiting the availability of labels in function approximation tasks, DLRSE enables the network to focus on high-loss regions while maintaining a comprehensive view of the function space. Furthermore, DLRSE augments the dataset in areas with high loss, allowing the network to gain a more complete understanding of the loss landscape during training, thereby facilitating the discovery of more optimal weights.

Table I demonstrates that, with 32 hidden neurons, the network trained with DLRSE achieves superior approximation accuracy compared to [12], while significantly reducing the number of parameters (P) to only 97 (64 in the first layer and 33 in the second layer), representing merely 1.67% of the parameters used in [12]. Ablation studies further validate the essential synergistic role of $\mathbf{x_a}$ and $\mathbf{x_d}$ in the success of DLRSE. It is noteworthy that in terms of weight efficiency $\left(\frac{1}{R_{MAE}\cdot P}\right)$, the average efficiency for the network trained via conventional methods is approximately 1.7×10^{-4} [12]. A two-layer ReLU neural network trained by naive training strategies, (i.e., without $\mathbf{x_d}$) achieves a similar value of 1.9×10^{-4}. In contrast, the proposed DLRSE-trained network attains a markedly higher value of 1.6×10^{-2}, attributable to the superior quality and diversity of training data dynamically supplied by DLRSE. This significant enhancement in weight efficiency indicates that the network can maintain equivalent approximation accuracy with a substantially more compact architecture, thereby reducing both the area and energy consumption of the nonlinear operator implemented using the trained network.

III. PROPOSED PIPO-UNPE

The schematics of the proposed PIPO-UNPE are shown in Fig. 2(a) and Fig. 2(b). The PIPO-UNPE comprises two cascaded layer circuits corresponding to the proposed two-layer network. In the neuron circuits of each layer, the R-PPCS array serves as the core unit responsible for storing weights and performing MAC operations. The implementation of R-PPCS is depicted in Fig. 2(c) and Fig. 2(d), which introduce two variants: R-PPCS-A and R-PPCS-B. Both variants consist of RRAM, a current buffer (CB), and current steering switches. The CB plays a critical role by providing the clamping voltage V_{BL} to read the RRAM conductance and transmitting the current I_G through the RRAM to the output of CB. The computation mechanism of R-PPCS is illustrated in Fig. 2(e) and Fig. 2(f). One R-PPCS can perform a one-quadrant (1Q) dot product. Subsequently, results from multiple R-PPCSs can be accumulated and stored by connecting them to the

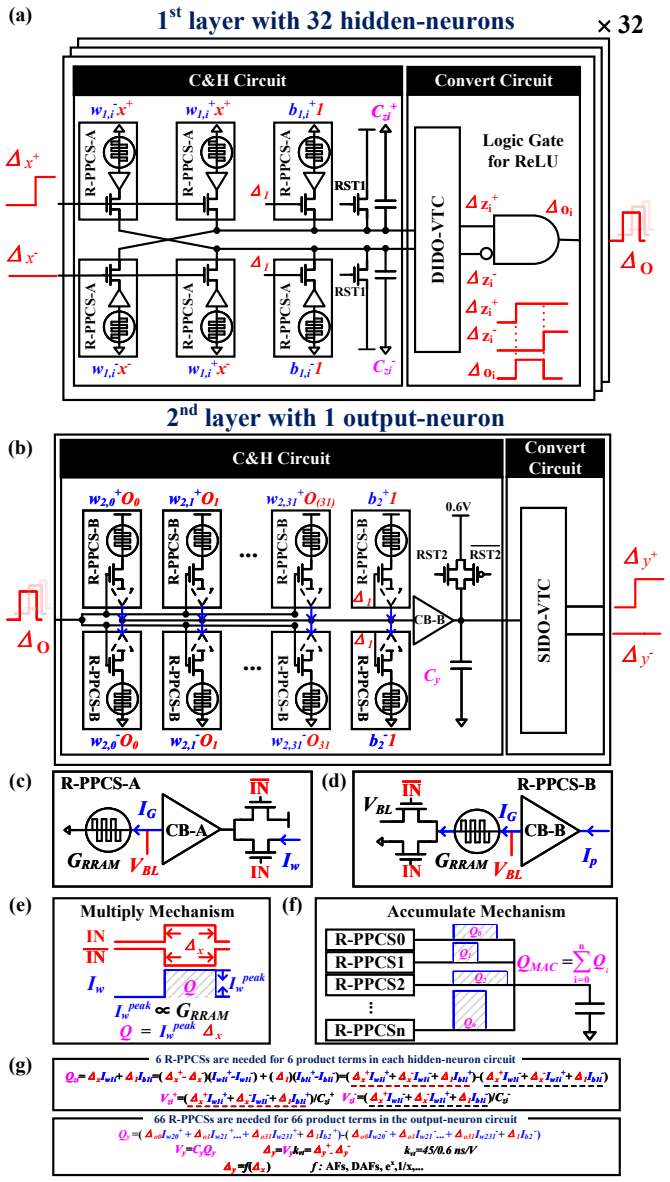


Fig. 2. Overview of the PIPO-UNPE System: (a)-(b) the circuit implementation of each layer in the proposed network, (c)-(d) diagrams of two types of R-PPCS , (e)-(f) the MAC mechanism and (g) the computation principles of each neuron circuit.

same capacitor. The difference between R-PPCS-A and R-PPCS-B lies in their array configurations. An R-PPCS-A array is composed of multiple parallel independent R-PPCS-As, whereas an R-PPCS-B array consists of multiple pairs of RRAM-switches sharing a single CB-B. Fig. 2(g) illustrates the computation principles of the PIPO-UNPE.

The functional waveform of the PIPO-UNPE is shown in Fig. 3. Both the hidden-neuron and output-neuron circuits follow an identical operation process, consisting of three phases: Charge & Hold (C&H), Convert, and Reset. During the C&H phase, the R-PPCSs charge the integration capacitors, driven by the input pulse width and the constant bias pulse width (Δ_1). In the Convert phase, computing results stored

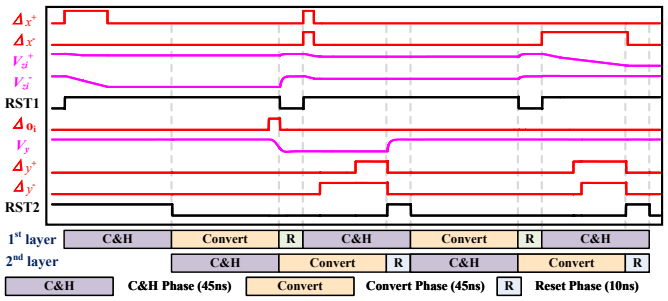


Fig. 3. The functional waveform of the PIPO-UNPE

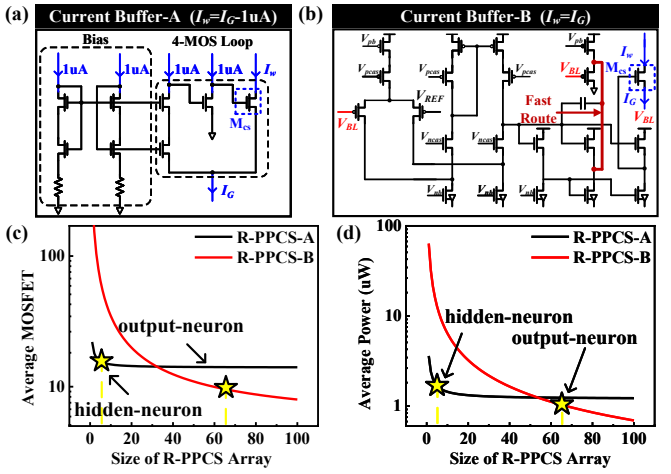


Fig. 4. (a)-(b) Circuit diagrams of CB-A and CB-B. (c) Average MOSFET overhead and (d) average power consumption for CB-A and CB-B arrays across various size.

in the integration capacitors are converted into differential pulse widths using the Differential-IN-Differential-Out VTC (DIDO-VTC) and the Single-IN-Differential-Out VTC (SIDO-VTC). The integration capacitors are reset during Reset phase to prepare for the next C&H phase. It is noteworthy that in order to achieve optimal computational accuracy and hardware efficiency, significant differences exist between the choice of R-PPCSs, bipolar harge storage strategies, and VTC implementations in both hidden-neuron circuit and the output-neuron circuit due to their different scales of MAC operations. These distinctions will be discussed in detail subsequently.

A. R-PPCS-A VS R-PPCS-B

The primary distinction between R-PPCS-A of the hidden-neuron circuit and R-PPCS-B of the output-neuron circuit lies in their respective CBs. Fig. 4(a) and Fig. 4(b) illustrate the proposed CB-A and CB-B respectively. The main difference between CB-A and CB-B is the implementation of the M_{CS} gate voltage controller. CB-A features a four-transistor loop to ensure stable I_G transmission, whereas CB-B employs a two-stage amplifier with a fast response path to handle dynamic MAC current effectively. Although CB-B is individually more complex than CB-A, this does not imply that an R-PPCS-B array incurs higher hardware costs compared to an R-PPCS-A

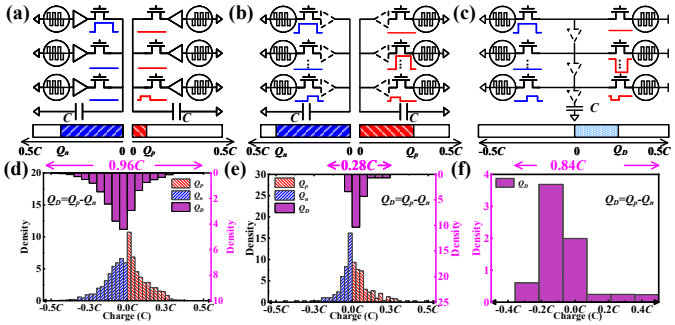


Fig. 5. (a)-(b) Differential capacitor pairs in the hidden-neuron and output-neuron circuits. (c) Pre-charge differentiation in the output-neuron circuit. (d)-(e) Overview of charge storage in the hidden and output-neuron circuits using differential capacitor pairs. (g) Situation of Q_D in the output-neuron circuit with pre-charge differentiation.

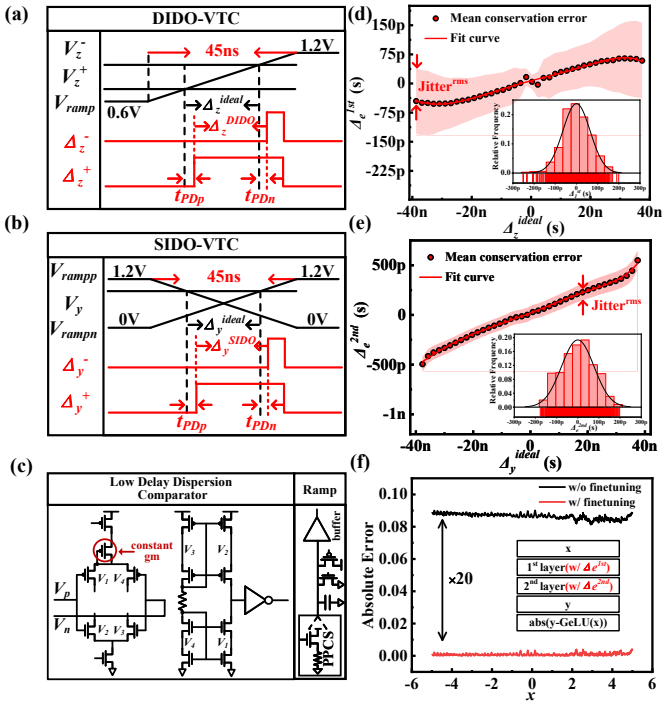


Fig. 6. (a)-(b) Operating mechanisms of DIDO-VTC and SIDO-VTC. (c) Circuit diagram of low delay dispersion comparator and ramp generator. (d)-(e) Conversion errors of DIDO-VTC and SIDO-VTC, where $\Delta_e^{1st} = \Delta_z^{ideal} - \Delta_z^{DIDO}$ and $\Delta_e^{2nd} = \Delta_y^{ideal} - \Delta_y^{SIDO}$. (f) Retrain the network to compensate for the precision loss introduced by VTCs.

array, since multiple R-PPCS-Bs can share the same CB-B. As demonstrated in Fig. 4(c) and Fig. 4(d), simulation results indicate that R-PPCS-B is particularly well-suited for constructing large-scale R-PPCS arrays, which are necessary for the output-neuron circuit (sixty-six 1Q products). Conversely, R-PPCS-A is more appropriate for hidden-neuron circuits (six 1Q products) that require smaller-scale R-PPCS arrays.

B. Bipolar Charge Storage Method

Differential capacitor pairs are commonly used for bipolar charge storage in TD-IMC systems. As demonstrated in Fig. 5(a) and detailed in Fig. 5(d), the utilization of this technique within the hidden-neuron circuit yields a charge storage efficiency of approximately 92%. However, as shown in Fig. 5(b) and Fig. 5(e), this efficiency decreases to 48% within the output-neuron circuit due to common-mode charges ($Q_D = Q_p + Q_n$) being stored by the capacitors instead of differential-mode charges ($Q_C = Q_p - Q_n$), restricting the signal swing across capacitors and subsequently reducing output precision. To address this concern,as portrayed in Fig. 5(c), we propose to use opposing currents in the R-PPCS-B array within the output-neuron circuit to facilitate charge differentiation prior to reaching the capacitor, capitalizing on the CB-B's ability to transmit bidirectional currents. Following this modification, as demonstrated in Fig. 5(f), the storage efficiency significantly improves to reach 84%.

C. SIDO-VTC VS DIDO-VTC

Fig. 6(a) and Fig. 6(b) show the mechanisms of the DIDO-VTC and the SIDO-VTC. The DIDO-VTC converts differential voltages (V_z^+ and V_z^-, 0.6V-1.1V) to differential pulse widths; SIDO-VTC converts V_y (0.1V-1.1V) similarly. Fig. 6(c) depicts the ramp wave generator and comparator utilized by both VTCs. By employing constant-gm technology [15], we significantly reduce the conversion error (Δ_e), which is primarily caused by propagation delay dispersion between p-type and n-type transistors ($t_{\mathrm{PDp}} - t_{\mathrm{PDn}}$). The ramp generator's structure, analogous to the R-PPCS array, ensures stable slopes and enhances conversion accuracy. Fig. 6(d) and 6(e) present the conversion errors under Monte Carlo simulations for the DIDO-VTC and SIDO-VTC, respectively. The conversion errors in both VTCs can be decomposed into two components: a systematic bias arising from inherent non-idealities in the circuit structure, manifested as the average conversion error, and random fluctuations due to process variations, resulting in jitter. While further reducing conversion errors can mitigate accuracy loss caused by circuit imperfections, compensating for these errors through traditional circuit design introduces additional hardware overhead. Alternatively, addressing this issue at the software level using hardware error-aware training strategies is more cost-effective. By leveraging the strong adaptability of neural networks, a network model further trained with hardware errors can effectively accommodate and compensate for these errors.

In this work, we focus on compensating for the relatively stable average conversion error, which dominates the overall error profile. Specifically, we performed a sixth-order polynomial approximation of the average conversion error and incorporated the approximated error curves of the first and second layers into the trained model as additive errors. As demonstrated in Fig. 6(f), using the GeLU function as an example, the approximation accuracy of the PIPO-UNPE improves nearly 20-fold after fine-tuning compared to its pre-tuned performance.

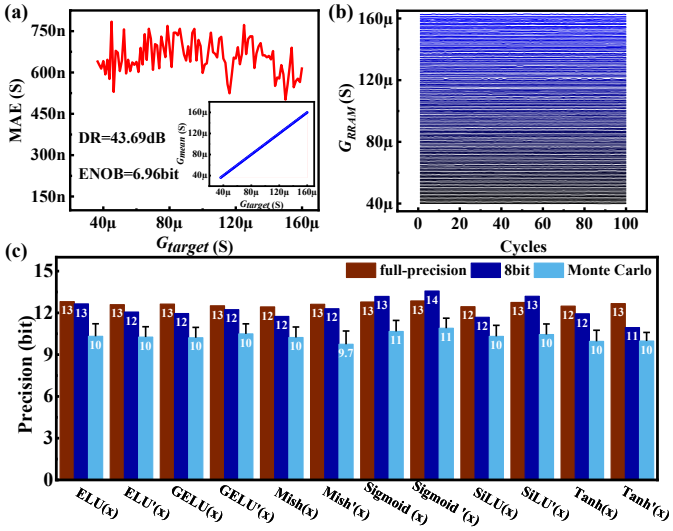


Fig. 7. (a) Random target programming results for 512 RRAMs. (b) The relaxation of 128 programmed states, in which each conductance level provided by 20 random RRAMs. (c) Approximation precision of PIPO-UNPE under hybrid simulation, calculated as (20lg(10/MAE)-1.76)/6.02.

IV. Results and Discussion

To bridge the performance gap between simulation and hardware, we adopted a hybrid simulation strategy in our study. The RRAM conductance values for the PIPO-UNPE were sourced from a commercial RRAM array, and the circuit design and simulation were based on a 130 nm technology library. For device performance evaluation, we conducted random programming experiments to explore the continuous programming characteristics of the RRAM in PIPO-UNPE. As shown in Fig. 7(a), a total of 512 valid RRAMs were randomly programmed to target conductance values within a precision requirement of ±0.5% across the conductance range from 40 uS to 160 uS. The RRAM exhibited a dynamic range (DR) of 43 dB, corresponding to approximately 7 bits of accuracy. Utilizing the results from the random programming experiments, we programmed the RRAMs linearly into 128 conductance states. As shown in Fig. 7(b), the conductance states after programming exhibited minimal overlap over 100 read cycles, indicating excellent state stability within the selected programming range, where the RRAM operates in the high-conductance state. In subsequent experiments, the measured RRAM conductance data were directly applied to the circuit simulation to reliably evaluate the performance of

TABLE II
Model Accuracy with Different Activation Functions

Quantization	MobileViT	ConvNeXt	YOLOv8	EfficientNetV2
8-bit Weight FP Activation	77.7%	82.4%	48.5%	83.2%
8-bit Weight 8-bit Activation	76.1%	81.1%	47.2%	82.4%
8-bit Weight PIPO-UNPE	77.4%	82.2%	48.7%	83.1%

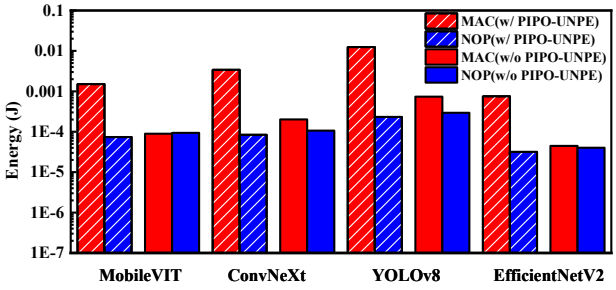


Fig. 8. Energy efficiency enhancements of TIMELY caused by PIPO-UNPE, with the digital nonlinear module utilizing NN-LUT scaled down to 130 nm [5].

PIPO-UNPE. For function testing, we designed approximation tasks for six types of activation functions and their derivatives, totaling twelve functions, to assess the nonlinear computation performance of PIPO-UNPE. Specifically, for each function, we first trained the two-layer neural network embedded within PIPO-UNPE using DLRSE to approximate the target function. Then, we quantized the trained full-precision model to 8 bits to align with the 7-bit maximum precision achievable by a single RRAM cell. Finally, the conductance data sequences corresponding linearly to the quantized weights were loaded into the circuit model for Monte Carlo simulation testing. As illustrated in Fig. 7(c), PIPO-UNPE achieved an average precision of 9.5 bits, representing a 3.4 bits precision loss compared to the full-precision model.

Notably, when applied to advanced network models, PIPO-UNPE meets the computational requirements of the activation functions used in these models. As shown in Table II, we tested four models—MobileViT, ConvNeXt, YOLOv8, and EfficientNetV2—using full-precision (FP) activation functions, 8-bit quantized activation functions, and activation functions implemented with the proposed PIPO-UNPE. The experimental results indicate that PIPO-UNPE performs comparably to full-precision activation functions, with an average accuracy loss of only 0.1%, which is significantly lower than the 1.25% degradation observed with 8-bit quantized activation functions.

Furthermore, integrating PIPO-UNPE into TD-CIM accelerators can significantly enhance their energy efficiency. For instance, in the TD-CIM accelerator TIMELY [13], the absence of nonlinear pulse-width computing resources necessitates transmitting the pulse widths output by the MAC array to the digital domain via TDCs to perform all computations required by the model. In contrast, PIPO-UNPE enables direct nonlinear processing of pulse widths, allowing a substantial portion of the MAC outputs to remain in the analog domain and thereby reducing the need for digital-to-analog conversions. As illustrated in Fig. 8, incorporating PIPO-UNPE improves the energy efficiency of TIMELY by factors of 9.5×, 19×, 25×, and 12× for MobileViT, ConvNeXt, YOLOv8, and EfficientNetV2, respectively.

The power and area breakdowns of the PIPO-UNPE design are shown in Fig. 9. The total power consumption of PIPO-UNPE is 912 uW, and the total area is 10,516 um^2. Table

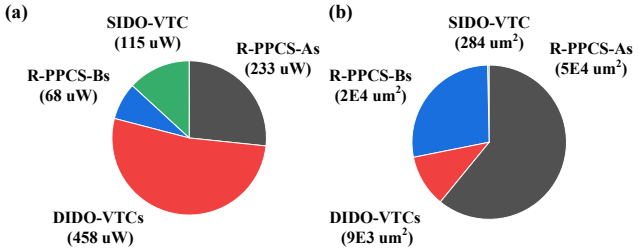


Fig. 9. Breakdown of the PIPO-UNPE: (a) power and (b) area.

TABLE III
COMPARISON WITH PREVIOUS DIGITAL AND ANALOG UNPE

	GEBA [4]	NN-LUT [5]	NACU [6]	ReAFM [7]	RAM-NAF [8]	PIPO-UNPE
Technology (nm)	28	7	28	20	65	130
Operation Mode	Digital	Digital	Digital	Digital	Time	Time
Functions (#)	4	4	5	10	3	12
Throughput (NOPS)	12.5M	360M	33M	40M	78M	10M
Power (mW)	9	0.025	1.035	N/A	0.168	0.912
Energy Efficiency (GNOPS/W)	1.4	1440	31	N/A	464	11
Normalized EE† (GNOPS/W)	0.71	8.1	2.1	N/A	7.2	11
Shareable	Yes	Yes	Yes	Yes	No	Yes
Reallocatable	No	No	No	No	No	Yes

† Normalized Energy Efficiency (EE): Power and throughput are scaled to a 130 nm process node, incorporating the TDC and DAC required for UNPE operations to ensure accurate EE evaluation [1], [16].

III presents a comparison table of PIPO-UNPE with other digital-analog UNPEs. Notably, after normalizing for size, PIPO-UNPE demonstrates a significant advantage in energy efficiency. Moreover, compared to other analog nonlinear operation schemes PIPO-UNPE offers two main advantages. First, itdirectly processes pulse-width signals that are easy to route, making it shareable among MAC arrays. Second, unlike traditional analog UNPEs that rely on polynomial approximations, the neural network-based PIPO-UNPE allows flexible reallocation of its computational resources as linear computation resources. This enables TD-CIM accelerators to dynamically distribute computational resources between linear and nonlinear operations according to the model's requirements.

V. CONCLUSION

In this work, we propose the PIPO-UNPE to address the lack of universal analog nonlinear operators in TD-IMC systems. The DLRSE effectively resolves the issue of low weight efficiency in traditional neural network training methods. Simulation results demonstrate that PIPO-UNPE not only exhibits high performance but also significantly enhances the energy efficiency of existing accelerators. Notably, because the computational resources of PIPO-UNPE align with those of existing accelerators, its integration is seamless and scalable—features not available in other proposed Nonlinear Processing Elements. We believe that PIPO-UNPE will strongly promote the development of flexible and computationally powerful TD-IMC accelerators.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 92364202 and Grant 62322412.

REFERENCES

[1] J.-M. Hung, T.-H. Wen, Y.-H. Huang, S.-P. Huang, F.-C. Chang, C.-I. Su, W.-S. Khwa, C.-C. Lo, R.-S. Liu, C.-C. Hsieh *et al.*, "8-b precision 8-mb reram compute-in-memory macro using direct-current-free time-domain readout scheme for ai edge devices," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 303–315, 2022.

[2] A. Whitcombe, S. Kundu, H. Chandrakumar, A. Agrawal, T. Brown, S. Callender, B. Carlton, and S. Pellerano, "22.3 a 76mw 40gs/s 7b time-interleaved hybrid voltage/time-domain adc with common-mode input tracking," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 392–394.

[3] C. Lu, B. Shi, and L. Chen, "Analogue circuit realization of a programmable sigmoidal function and its derivative for on-chip bp learning," in *IEEE APCCAS 2000. 2000 IEEE Asia-Pacific Conference on Circuits and Systems. Electronic Communication Systems.(Cat. No. 00EX394)*. IEEE, 2000, pp. 626–629.

[4] C. Ye and D. S. Jeong, "Geba: Gradient-error-based approximation of activation functions," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2023.

[5] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "Nn-lut: neural approximation of non-linear operations for efficient transformer inference," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 577–582.

[6] G. Baccelli, D. Stathis, A. Hemani, and M. Martina, "Nacu: a non-linear arithmetic unit for neural networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[7] X. Wu, S. Liang, M. Wang, and Z. Wang, "Reafm: A reconfigurable nonlinear activation function module for neural networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 7, pp. 2660–2664, 2023.

[8] C. Xie, Z. Shao, Y. Wu, C. Liu, M. Zhang, Y. Du, and L. Du, "Ram-naf: A reconfigurable analog module for nonlinear activation function computation in processing-in-memory," *Authorea Preprints*, 2024.

[9] S. Vlassis, F. Khateb, and G. Souliotis, "An on-chip linear, squaring, cubic and exponential analog function generator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 94–104, 2018.

[10] M. T. Abuelma'ati and A. Shwehneh, "A reconfigurable satlin/sigmoid/gaussian/triangular basis functions computation circuit," in *APCCAS 2006-2006 IEEE Asia Pacific Conference on Circuits and Systems*. IEEE, 2006, pp. 1172–1175.

[11] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova, "Nonlinear approximation and (deep) relu networks," *Constructive Approximation*, vol. 55, no. 1, pp. 127–172, 2022.

[12] X. Xu, Z. Jiang, G. Chen, Z. Li, G. Gong, and H. Lu, "General nonlinear function neural network fitting algorithm based on cnn," in *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)*. IEEE, 2022, pp. 1079–1082.

[13] W. Li, P. Xu, Y. Zhao, H. Li, Y. Xie, and Y. Lin, "Timely: Pushing data movements and interfaces in pim accelerators towards local and in time domain," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 832–845.

[14] B. Liu and Y. Liang, "Optimal function approximation with relu neural networks," *Neurocomputing*, vol. 435, pp. 216–227, 2021.

[15] M. Wang, T. L. Mayhugh, S. H. Embabi, and E. Sánchez-Sinencio, "Constant-g/sub m/rail-to-rail cmos op-amp input stage with overlapped transition regions," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 2, pp. 148–156, 1999.

[16] Z. Xuan and Y. Kang, "High-efficiency data conversion interface for reconfigurable function-in-memory computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 9, pp. 1193–1206, 2022.

UVLLM: An Automated Universal RTL Verification Framework using LLMs

Yuchen Hu[1,2], Junhao Ye[1,2], Ke Xu[1,2], Jialin Sun[1,2], Shiyue Zhang[1,2], Xinyao Jiao[1,2], Dingrong Pan[2], Jie Zhou[1,2], Ning Wang[3], Weiwei Shan[1,2], Xinwei Fang[4*], Xi Wang[1,2*], Nan Guan[3], Zhe Jiang[1,2*]
[1]School of Integrated Circuits, Southeast University, China [2]National Center of Technology Innovation for EDA, China
[3]Department of Computer Science, City University of Hong Kong, Hong Kong
[4]Department of Computer Science, University of York, UK

***Abstract*—Verifying hardware designs in embedded systems is crucial but often labor-intensive and time-consuming. While existing solutions have improved automation, they frequently rely on unrealistic assumptions. To address these challenges, we introduce a novel framework, UVLLM, which combines Large Language Models (LLMs) with the Universal Verification Methodology (UVM) to relax these assumptions. UVLLM significantly enhances the automation of testing and repairing error-prone Register Transfer Level (RTL) codes, a critical aspect of verification development. Unlike existing methods, UVLLM ensures that all errors are triggered during verification, achieving a syntax error fix rate of 86.99% and a functional error fix rate of 71.92% on our proposed benchmark. These results demonstrate a substantial improvement in verification efficiency. Additionally, our study highlights the current limitations of LLM applications, particularly their reliance on extensive training data. We emphasize the transformative potential of LLMs in hardware design verification and suggest promising directions for future research in AI-driven hardware design methodologies. The Repo. of dataset and code: https://github.com/SEU-ACAL/reproduce-UVLLM-DAC-25/.**

I. Introduction

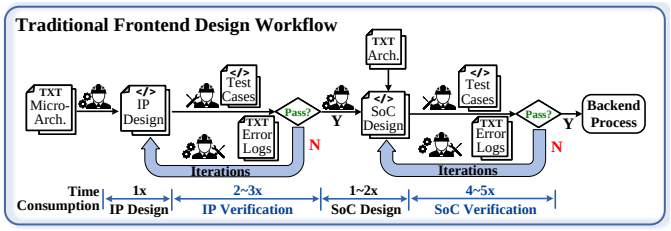
Hardware design verification in embedded systems remains heavily dependent on human expertise, making it a tedious and error-prone process that often incurs significant costs [1], as Fig. 1(a) illustrates. Originally developed for software [2]–[6], automated program repair (APR) uses automated tools to fix errors with minimum human intervention and is now being adapted for Hardware Description Language (HDL) design verification due to its potential to reduce human errors and verification costs.

APR systems, as depicted in Fig. 1(b), receive design codes and test cases, and attempt to enact targeted modifications with predefined templates to ensure all tests are passed. Innovations such as Cirfix [7], Strider [8], and RTLrepair [9] demonstrate the potential of APR to reduce the labor and time required for hardware design verification. However, these APR methodologies predominantly rely on fixed templates and focus on addressing functional error, limiting their scope and effectiveness of the repairs.

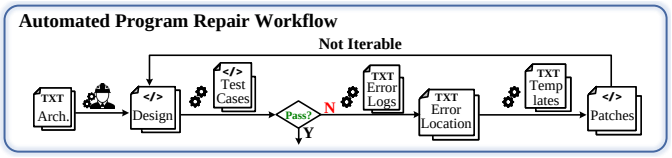
Fortunately, recent advancements in LLMs such as generating hardware code from natural language specifications [10]–[13], and debugging hardware designs for both functional and syntax errors [14]–[18], have significantly bridged this gap. However, existing solutions are still insufficient for hardware design verification due to their reliance on unreliable assumptions and lack of consideration for the limitations inherent in LLMs. Notably, while studies like [17] report high fix rates, our analysis reveals 10% of benchmarks pass testbenches unrepaired, and certain repairs remain unreliable due to incomplete test coverage. These results highlight the urgent need for more robust solutions suitable for real-world verification scenarios.

To conclude, the prevailing research on LLM-aided debugging tends to overly highlight capabilities of LLMs but neglects the crucial need for systematic testing and repair. Regarding this issue, we propose a comprehensive end-to-end hardware design verification framework, Universal Verification via Large Language Model (UVLLM),

*Corresponding author:
Zhe Jiang. Email: zhejiang.uk@gmail.com.
Xinwei Fang. Email: xinwei.fang@york.ac.uk.
Xi Wang. Email: xi.wang.sudo@gmail.com.



(a) Workflow of the Frontend Hardware Design.



(b) Workflow of the APR System.

Fig. 1: The frontend process is divided into initial design and verification phases, with verification accounting for more than 70% of the duration [1]. Advanced APR techniques are integrated to automate and accelerate the repair stage during verification phase (</>: RTL codes).

that integrates the established UVM with emerging and off-the-shelf methods. By embedding practical design assumptions and mitigating LLM uncertainties, UVLLM enables the first automated, systematic approach to hardware design verification, surpassing current LLM-based debugging solutions in robustness.

The main **contributions** of this paper are:

- **A comprehensive testing:** Utilizing the UVM, our approach enables flexible test modes and efficient coverage collection. Additionally, the reference models generated by LLMs provide a robust foundation for testing across diverse input scenarios.
- **An open-sourced tooling:** We have created an open-source toolset to enable the broad adoption of UVLLM, thereby easing its integration. These tools are publicly available at **https://github.com/SEU-ACAL/reproduce-UVLLM-DAC-25/**.
- **Extensive empirical validation:** We present an open-source error dateset derived from verified projects, containing 331 code instances with realistic errors across various modules, generated by our paradigm error generator. We will continue to organize and update this dataset periodically.
- **Demonstrated performance improvement:** UVLLM, incorporating GPT-4-turbo, advances verification automation with syntax and functional error fix rates of 86.99% and 71.92%, respectively, exceeding MEIC [17] in terms of repair rates and execution time under realistic scenarios. It achieves up to 48× speedup over manual debugging by experienced engineers.

Organisation. The structure of this paper is as follows: Section II outlines the fundamental concepts of UVLLM, while Section III delves into the details of UVLLM and their underlying reasons. Section IV assesses our framework and compares it with existing methods. Section V provides conclusions and make discussions.

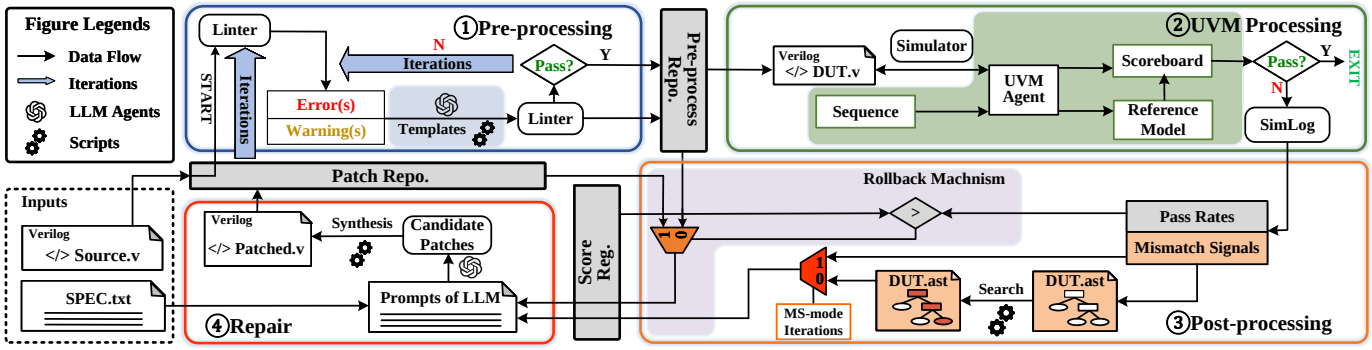


Fig. 2: The UVLLM Framework Overview: The process begins with the DUT and the Specification (Spec.), where the Spec. is used to generate a reference model. Initially, the DUT is pre-processed to eliminate syntax and focused timing-related errors (Step ①). Subsequently, the pre-processed code is tested under a UVM testbench (Step ②), and the log is then post-processed to extract relevant data (Step ③), which the debug agents use to generate candidate patches (Step ④). These codes and their pass rates are archived in the Repository (Repo.) and Register (Reg.) for future iterations.

II. UVLLM: An Overview

Designed to enhance the hardware design verification phase, UVLLM aids hardware developers in detecting and correcting common errors in RTL code. The UVLLM is applicable to various hardware environments, in this work, we illustrate the usage of UVLLM in Verilog. As depicted in Fig. 2, this framework combines traditional UVM with LLMs and assumes following inputs:

- **Design specifications** that outline the intended and expected behavior of the hardware component;
- **RTL codes** that contain the untested RTL code of the initial hardware design, i.e., Design Under Test (DUT).

Reflecting advancements in state-of-the-art (SOTA) research, it is noted that using LLMs for hardware design verification requires an iterative approach [17] and is more effective when LLMs are provided with detailed error information [19]–[21]. However, using LLMs still presents certain shortcomings, as current applications of LLMs struggle with reliability, applicability, and processing long code sequences. Additionally, the use of SOTA LLMs can be costly(e.g., the GPT-4-Turbo model, charges \$0.01 per $1K$ input tokens and \$0.03 per $1K$ output tokens [22]). To address these challenges, UVLLM introduces a cost-efficient, structured four-step process for verifying RTL code against design specifications, as illustrated in Fig. 2.

1) ***Pre-processing:*** Starting with raw RTL code as the input, this stage utilizes linters such as Verilator to pre-process the code, removing syntax errors and addressing timing-related functional errors using a combined LLM-script method. The output is syntax-correct DUT, setting a solid foundation for further functionality testing.

2) ***UVM Processing:*** Pre-processed DUT code is then tested against a pre-built UVM testbench to identify behavioral discrepancies. Outputs include detailed logs that either confirm alignment with the reference model or highlight deviations with specific signal values and test pass rates, facilitating targeted repairs.

3) ***Post-processing:*** Utilizing the UVM logs as input, this stage analyzes the logs to extract critical error data using a localization engine and the Abstract Syntax Tree (AST). The output isolates mismatch signals and specific error paths, preparing them for precise correction in the repair stage.

4) ***Repair:*** The final stage takes the design description, the DUT code and the detailed error information from the post-processing as input. Utilizing the information, the debug agents offer candidate patches to correct the errors. The repaired DUT code is then synthesized as the stage output for further iteration.

The termination conditions for the framework loop are: **1)** no errors are detected (**Success**), or **2)** the maximum number of iterations is reached (**Failure**). If any of the above conditions are met, the iteration stops. All history files are stored for reference.

Algorithm 1: Pre-processing DUT with Joint LLM-Script.

Input: DUT file F_D
Output: Pre-processed DUT file F_{Dprep}
Function PreproDUT(F_D) **:**
  repeat
    Log = Linter(F_{Dprep});
    $Errs$ = Match(Log, $Error$);
    $Warns$ = Match(Log, $Warning$);
    if $Errs$ **then**
      F_{Dprep} = GPT(F_{Dprep}, $Errs$);
    else if $Warns$ **then**
      $WarnTemps$ = Search($Warns$, $WarnList$);
      F_{Dprep} = Replace(F_{Dprep}, $WarnTemps$);
    end
  until $(Errs == \varnothing)\&(Warns == \varnothing)$;
  return F_{Dprep}
End Function

Modularization. UVLLM uses modular design which allows it to adapt to a wide range of verification scenarios by enabling the use of different tools based on the needs. For example, one can replace an LLM with a more advanced model or use a different linter if required. This flexibility is made possible by standard interfaces between the pipelines, which simplify the integration of different tools via adjustments to the API or by keeping a consistent format.

III. UVLLM: The Framework Pipeline

As for the operation of the framework, we introduce the joint LLM-Script pre-processing stage (Section III-A), processing stage with tests (Section III-B), post-processing stage for error location (Section III-C), followed by the discussion on the microsystems integrated with the LLM agents (Section III-D). At last, we present our effort for human-like error generation (Section III-E) for evaluation.

A. Pre-processing via the Linter

Following best practices, Verilator linting pre-processes code before UVM testing to catch functional defects. Static slicing and analysis identifies error sources without execution, while LLMs refine Verilog by fixing syntax (using error context) and flagging early semantic issues. This reduces downstream LLM debugging reliance—cutting both cost and iteration time.

Combined LLM-Script Pre-processing. To further reduce costs, as detailed in Algorithm 1, we employ a cost-effective strategy that combines LLMs with supplementary scripting to minimize unnecessary LLM usage. In the pre-processing stage, LLMs are only utilized to help identify and correct syntax errors. These models draw on their extensive training across diverse codebases, efficiently recognizing and amending a broad spectrum of common coding errors and integrating essential error information.

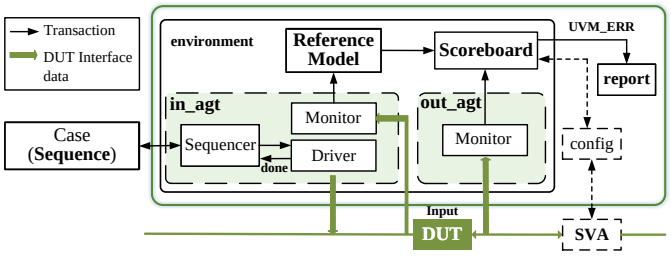


Fig. 3: Structure of the UVM [23], illustrating the key components and their interactions within a typical verification environment.

Additionally, the pipeline incorporates scripts designed to address specific warnings that, while not syntax errors, could lead to potential runtime issues, especially some timing-related ones. For instance, in combinational logic circuits, blocking assignments are typical, and Verilator issues warnings for using non-blocking assignments in this case. Through predefined templates, it is possible to systematically identify and modify issues where a non-blocking assignment "<=" should be replaced with a blocking assignment "=", ensuring the code adheres to expected timing behaviors.

The pre-processing stage iterates until all syntax errors and focused timing-related warnings are resolved. Integrating LLM agents with scripts establishes a robust foundation for subsequent verification stages, ensuring that the DUT encounters only functional errors.

B. UVM Processing

Recent studies, such as MEIC [17], validate LLMs' effectiveness in hardware debugging. However, these studies neglect the importance of testbench construction and employ finite test cases, which restricts the test coverage and leads to overfitting on specific cases. This limited scope significantly undermines the general applicability of the results, as evidenced by a **10%** reduction in the actual fix rate reported by MEIC due to many error instances escaping detection. To overcome these limitations, the UVM framework, depicted in Fig. 3, is employed as the testbench for UVLLM to verify RTL codes, due to its robust support for flexible and complete testing modes.

UVM Construct. The UVM framework provides a sophisticated formal verification structure that extends beyond basic testbenches. It integrates components such as agents, environments, sequencers, drivers, and monitors, with each agent comprising a sequencer, driver, and monitor that interact directly with the DUT. The sequencer organizes transactions derived from sequences that emulate real-world operations, which the driver then converts into pin-level stimuli on the DUT. A critical component is the scoreboard, which systematically compares observed results against expected outcomes to verify DUT. Furthermore, UVM facilitates various test modes and coverage collection strategies to enhance the completeness of the verification process, effectively identifying discrepancies and potential failures.

Reference Model Generation. In UVM, reference models play a crucial role in verifying complex designs, such as those used in digital signal processing and cryptography, by providing high-level abstractions of the DUT. These models enhance simulation accuracy and efficiency, contributing to a more streamlined verification process. Traditionally, C/C++ is preferred for reference models in industry due to its seamless integration with SystemVerilog via Direct Programming Interfaces (DPI), which accelerates verification cycles [24]–[26]. LLMs now enhance this workflow: leveraging open-source datasets, they generate adaptable C/C++ reference models that dynamically evolve with verification needs. These LLM-aided models improve high-fidelity simulation and design validation, reducing manual effort in maintaining complex abstractions.

Extensibility. The extensibility of the UVM is particularly evident when considering the integration of automated assertion generation. UVM's structured, modular framework for verification is optimally configured to incorporate advanced enhancements such as AI-driven assertions, which can systematically verify that the design behaves as expected across various protocols like APB (Advanced Peripheral Bus) and AHB (Advanced High-Performance Bus) [27].

Algorithm 2: Post-processing with Localization Engine.

Input: DUT file F_D, UVM log L_{UVM}, reference waveform W_R, simulation waveform W_S, iterations $Iter$
Output: Error information $ErrInfo$

Function ErrChk(L_{UVM}, W_S) **:**
  /* MT: $Mismatch\ Timestamp$ */
  /* MS: $Mismatch\ Signals$ */
  /* IV: $Input\ Values$ */
  MT, MS = getMismatch(L_{UVM}, PAT_{MS});
  if *MS* **then**
    IV = getInputValue(W_S, MT);
  end
  return MT, MS, IV
End Function
Function ErrInfoFetch(F_D, L_{UVM}, W_R, W_S, $Iter$) **:**
  /* SL: $Suspicious\ Code\ Lines$ */
  MT, MS, IV = ErrChk(L_{UVM}, W_S);
  for $ms \in MS$ **do**
    DFG = getDFG(F_D, ms);
    $SL = SL \bigcup$ traverse(DFG, IV);
    if *detectSignal*(FL, s) *and* $s \notin MS$ **then**
      $MS = MS \bigcup \{s\}$;
    end
  end
  $ErrInfo = (Iter < TH)\ ?\ MS : SL$;
  return $ErrInfo$
End Function

C. Post-processing via Localization Engine

Current LLM-aided verification methods tend to use minimally processed logs as inputs, which are often low in information density, thereby diminishing the efficiency of LLMs in diagnosing and fixing errors. To address this, we adopted time-aware dynamic error localization [8] to extract more concrete and high-value information from these logs with methods. This method, tailored for HDL environments, surpasses the static localization method described in Section III-A by providing greater precision and temporal sensitivity.

The localization engine uses dynamic analysis with temporal tracking to pinpoint UVM log signal mismatches, driving DFG-based dynamic slicing (Algorithm 2). To optimize LLM token efficiency, UVLLM employs tiered information extraction: initial prompts use mismatch signals as error indicators, but if repairs fail, suspicious execution paths, which are identified via temporal analysis, are added to refine diagnostics. This hybrid strategy ensures precision without overloading the LLM, leveraging signal discrepancies and runtime context to resolve complex hardware verification failures. This approach focuses on actual execution paths in operation, leading to more precise identification of errors.

Rollback Mechanism. During the development of UVLLM, a *Rollback mechanism* was implemented to address the issue of inaccuracies in LLM outputs, often termed "hallucination" [28]–[32]. This feature is crucial for preventing the accumulation of errors across iterations due to reliance on flawed candidate repairs. Despite previous studies utilizing LLMs as reward models to assess repair quality [17], [33], there is a lack of robust quantitative metrics to effectively measure the correctness of these modifications. This gap can lead to inefficiencies in the rollback process, potentially triggering false positive rollbacks.

Within the UVM framework, the quality of Verilog code iteration is evaluated using a scoreboard that assigns the test pass rate. We assume that higher scores correlate with fewer errors and better functionality. The Rollback mechanism functions by preserving a

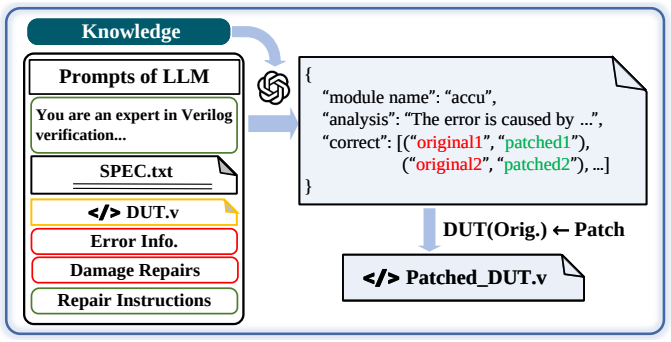


Fig. 4: Input and Output Formats for LLM Agents: The input format is structured for various agents with minor prompt modifications. The output is provided in JSON format, featuring original-patch pairs.

history of code versions and their scores. If a new iteration scores lower than a previous one, indicating a decline in code quality, the mechanism reverts to the highest-scoring version, as illustrated in Fig. 2. The alterations that led to the decrement in score are thus recorded as "damage repairs", which is utilized in Fig. 4.

D. Repair Agent

The LLM agent functions as an adept RTL repair expert, leveraging three key inputs: the design specification, which outlines intended functionality and port definitions; RTL code; and error information. To enhance the repair process and facilitate iterative improvements, the system incorporates "damage repairs" as an additional input, crucial for preventing the recurrence of unsuccessful corrections when the rollback mechanism is activated. In a multi-agent setup, specific modifications to the prompts for each agent address different aspects of the debugging process, enabling a more nuanced and thorough error analysis in the RTL code. The primary prompts that guide the debugging activities of these agents are illustrated in Fig. 4, emphasizing a tailored approach to error resolution.

Formalizing agent's outputs. It's commonly noticed that LLM-generated responses tend to include detailed explanations during debugging, which can clutter the main objective of code debugging. In light of our observations, it becomes evident that LLMs exhibit enhanced debugging capabilities with superior reasoning process. To prevent the inclusion of irrelevant details and erase the hallucination during the iterative cycle, a method for distilling the responses generated by the LLM is adopted. Utilizing the Structured Outputs method enables adherence to JSON Schema [34], thereby ensuring that responses are consistently formatted according to predefined structures. By requiring the response to be in JSON format and to contain an element labeled "correct" which consists of pair of wrong codes and right codes, the code sections accentuated in Fig. 4 are refined and carried into the subsequent iteration.

E. Benchmark Generation

The incidence of errors in module code is closely related to specific attributes such as code length and functional complexity [35]–[37]. To evaluate the efficacy of the verification methodology, a well constructed evaluation dataset was developed through a systematic selection of samples from validated open-source datasets, representing a diverse range of codebases. These samples were then deliberately infused with typical errors to simulate real-world coding mistakes.

In design bases combining both commercial and open-source IPs, a comparative analysis was performed on two consecutive versions of the code: one immediately before and another after code repository commits. This analysis focused on identifying discrepancies and documenting changes made during the design process, effectively highlighting the differences between pre and post-commit versions. These error-modification pairs, detailed in Table I, were crucial

TABLE I: Part of common Verilog errors in real-world designs.

Types	Error	Symptoms
Declare	Type Misuse	output **reg** [8:0] result; output [8:0] result;
	Bitwidth Misuse	reg **[8:0]** count; reg **[7:0]** count;
Assignment	Operator Misuse	always @(*) result = a **+** b; always @(*) result = a **-** b;
	Variable Name Misuse	assign r1 = **r1_temp**; assign r1 = **r2_temp**;
	Value Misuse	if (rstn) data = **32'b0**; if (rstn) data = **32'b1**;
Condition	Wrong Judgment Value	for(i = 0; **i < 7**; i ++) begin ... end for(i = 0; **i < 15**; i ++) begin ... end
	Wrong Sensitivity	always@(posedge clk **or negedge rstn**) ... always@(posedge clk) ...
Port	Port Mismatch	mod mod1(.a(a), .b(b), .in_bd(**{bdg, 1'b1}**)); mod mod1(.a(a), .b(b), .in_bd(**1'b1**));

for developing prompts for LLM and for terms used in pattern matching, showcasing common human-made errors such as misuse of assignments and the mismatch port in instantiation. This approach showcases common human-made errors, such as misuse of assignments and mismatches in port instantiation, and provides a crucial benchmark for evaluating the verification effectiveness.

IV. Evaluation

This section presents our experimental setup, evaluation metrics research questions, and discussions to the results.

Setup. In our experiment, we employed LLM agents via the OpenAI API, with GPT-4-turbo as the default model. An evaluation benchmark was then constructed using the extensively verified RTLLM dataset [38], which encompasses a diverse array of real-world errors. The initial code's ability to pass the compiler is indicated by a syntax error or functional error. We utilized a range of simulation tools including VCS [39], Iverilog [40], Modelsim [41], Yosys [42], and the linting tool Verilator [43] to ensure comprehensive verification and analysis. We set the threshold of iterations to 5, as the improvement is hardly observed after that. All experiments were conducted on an AMD EPYC 7763 2.45GHz CPU. For each instance, we asked LLMs for 5 times to reduce the randomness of the response.

A. Evaluation Metrics

Recent work [44], [45], tended to use pass@k metrics to assess functional correctness. For each problem in the problem set, k code samples are generated at a time, and the problem is considered solved if any sample passes the simulation test.

Hit Rate (HR). Specifically, our framework quantifies effectiveness using Hit Rate (HR) [46]. For erroneous code θ_i and its corrected version θ_i^*, we evaluate a set of test cases $\{(x_i^1, y_i^1), \ldots, (x_i^m, y_i^m)\}$. The corrected code θ_i^* must produce the correct output y_i^j for each input x_i^j, ensuring all cases pass. That is, $\bigwedge_{j=1}^{m} a_{\theta_i^*}\left(x_i^j\right) = y_i^j$. The overall rate for n corrcted versions is calculated as:

$$\mathbf{HR} = \sum_{i=1}^{n} \frac{\bigwedge_{j=1}^{m} \left[a_{\theta_i^*}\left(x_i^j\right) = y_i^j\right]}{n} \times 100\% \tag{1}$$

HR measures the proportion of instances resolved under test cases.

Fix Rate (FR). To overcome limitations in test coverage, our framework includes Fix Rate (FR), which involves independent expert validation of the proposed fixes θ_i^*. After expert review, if the fix is confirmed effective across additional scenarios, it is designated as $\hat{\theta}_i^*$:

$$\mathbf{FR} = \sum_{i=1}^{n} \frac{\hat{\theta}_i^*}{n} \times 100\% \tag{2}$$

FR reflects the framework's repair effectiveness in broader conditions.

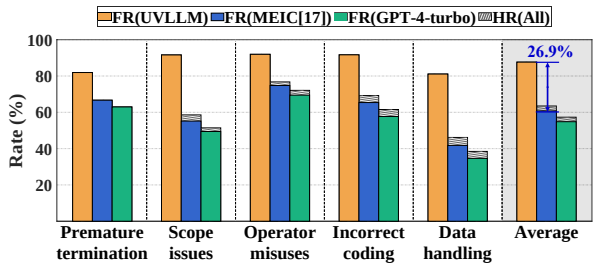


Fig. 5: HR vs. FR in Syntax-Error Verification with Different Methods [17]. The differences between HR and FR are shaded.

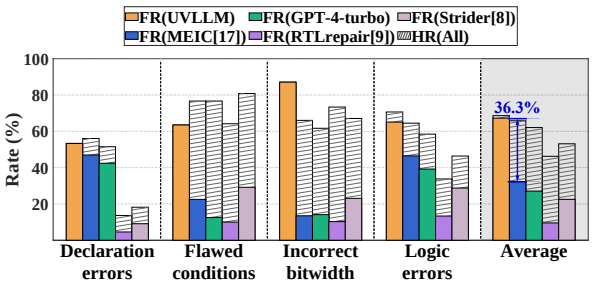


Fig. 6: HR vs. FR in Functional-Error Verification with Different Methods [8], [9], [17]. The differences between HR and FR are shaded.

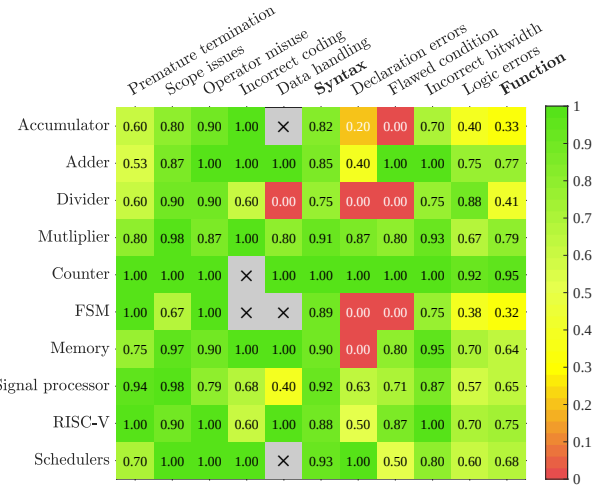


Fig. 7: Heat map result for FR. The symbol "×" represents an error that could not be imposed due to the limitations of the specific module structure. Syntax and Function represent the weighted mean of the FR for syntax errors and function errors, respectively.

Execution Time. In evaluating the framework's performance, this paper emphasizes execution time as a critical metric, defined as the time interval from the initial design input into UVLLM to the completion of the final code output.

B. Results and Discussions

Result 1:The UVLLM framework's effectiveness in enhancing hardware design verification is demonstrated through a comparative evaluation of Fix Rates (FRs) among various methodologies, as depicted in Figures 5 and 6. This analysis encompasses traditional script-based approaches [8], [9], current LLM-aided methods [17], and a baseline incorporating GPT-4-turbo for benchmark repairs. Across all categories of errors—both syntax and functional—the UVLLM framework consistently achieves higher FRs.

Specifically, for syntax errors, the UVLLM framework records an average FR of 87.6%, representing a significant 26.9% improvement over the second performing method, MEIC. For functional errors, UVLLM continues to surpass other methods, registering a FR of 67.3%, which is a 36.3% enhancement relative to MEIC. Notably, in the case of *Incorrect bitwidth* as illustrated in Fig. 6, UVLLM's FR is more than double that of the next best method, RTLrepair. Moreover, even in less favorable scenarios, such as *Declaration errors*, UVLLM's FR remains approximately 5% above that of MEIC, its nearest competitor. These results clearly demonstrate that UVLLM consistently outperforms other methods, achieving significantly higher FRs across diverse scenarios.

Result 2: The evaluation of the UVLLM framework's capability to ensure that repaired code adheres to specification requirements is conducted through an examination of the correctness of repaired code. Although many methods achieve high hit rates (HRs), their fixes often overfit to specific input-output (IO) pairs, revealing a discrepancy between HRs and FRs.

Figures 5 and 6 illustrate this disparity; the shaded areas represent the deviation between HR and FR, highlighting the failure of these methods to detect numerous errors, which leads to false negatives and insufficient coverage. Specifically, for syntax errors, UVLLM demonstrated no deviation across all scenarios, while deviations for other methods were observed in 4 out of 5 scenarios with an average of 5% variations, confirming the achievement of high coverage for syntax errors. In contrast, for functional errors, the deviations for other methods were notably higher (all above 30%) whereas UVLLM maintained a minimal deviation of only 1.4%. Notably, UVLLM had a maximum deviation of just 5.6% for *Logic errors*, while the other methods displayed deviations exceeding 40% for *Flawed conditions*.

These results suggest that while other methods struggle to achieve comprehensive coverage for functional errors, UVLLM effectively mitigates this limitation. These findings indicate that UVLLM significantly enhances the practicality of hardware design verification by integrating formal verification processes, to meet the specification requirements to the greatest extent possible.

Result 3: The UVLLM framework's repair capabilities across a diverse range of hardware modules were evaluated by analyzing the FRs of 27 common modules, each injected with nine distinct types of syntax and functional errors. These modules were categorized into ten representative types, such as *adders*, *counters*, and *FSMs*, to establish a comprehensive benchmark for evaluating the framework's generalization in various verification scenarios.

As illustrated in Fig. 7, where the framework's FRs were depicted using color coding, UVLLM exhibited exceptional adaptability, achieving robust FRs in simpler modules like *counters*. For instance, the FRs for syntax errors and functional errors in these modules reached 100% and 95%, respectively. In contrast, the FRs were lower in more complex modules, such as *FSMs*, with FRs for syntax errors and functional errors at 89% and 32%, respectively. This indicates that repairing more complex designs remains challenging.

Across the same types of module, syntax errors consistently exhibited higher FRs than functional errors, reflecting UVLLM's proficiency in addressing syntactic issues. This advantage stems from the extensive training of the LLM on a substantial corpus of HDL code data, enhancing its syntactic understanding. Additionally, the compiler and linter contribute detailed localization information that aids in the repair of syntax errors.

Overall, the framework achieved an FR of 86.99% for syntax errors and 71.92% for functional errors, representing its reliability across diverse modules and error scenarios.

Result 4: The UVLLM framework's evaluation primarily focuses on how its distinct stages contribute to the fix rate and execution time during the repair operation, providing insights into each segment of the verification process.

TABLE II: Performance comparison of segmented approach across common modules with various error instances.

Types	Pre-processing[1]		Repair in MS Mode		Repair in SL Mode		UVLLM[2]		MEIC [17]		Speedup
	$FR/\%$	T_{exec}/s	$FR/\%$	T_{exec}/s	$FR/\%$	T_{exec}/s	$FR/\%$	T_{exec}/s	$FR/\%$	T_{exec}/s	
Arithmetic[3] s[4]	69.93	8.30	13.07	5.60	1.31	0.30	84.31	14.20	62.30	197.29	**13.89x**
Control s	80.91	7.23	8.18	3.38	0.00	0.00	89.09	10.61	63.64	129.02	**12.61x**
Memory s	60.00	10.29	28.33	5.14	0.00	0.00	88.33	15.43	54.55	147.17	9.53x
Miscellaneous s	79.65	8.31	7.67	4.77	1.18	0.39	88.50	13.47	66.67	62.67	4.65x
Syntax	**74.72**	8.49	11.29	5.06	0.98	0.27	**86.99**	**13.83**	**62.99**	**134.95**	9.76x
Arithmetic f	30.26	3.63	33.33	11.27	2.63	0.64	66.23	15.54	40.53	257.28	**16.56x**
Control f	29.93	3.33	30.61	9.37	5.44	0.84	65.99	13.54	10.91	163.96	**12.55x**
Memory f	25.00	4.35	58.33	11.22	3.33	0.87	86.67	16.44	22.73	256.49	**15.60x**
Miscellaneous f	21.25	3.86	49.06	11.54	5.63	0.90	75.94	16.30	40.07	65.37	4.01x
Function	25.96	3.82	**41.46**	11.19	4.50	0.78	**71.92**	**15.79**	**34.57**	**191.76**	**12.14x**
Overall	**51.27**	**6.16**	**25.80**	**7.79**	**2.68**	**0.49**	**79.75**	**14.77**	**52.14**	**153.84**	**10.42x**

[1] The repair operation of UVLLM comprises three stages: Pre-processing, Repair in Mismatch Signal (MS) Mode, and Repair in Suspicious Line (SL) Mode. The columns labeled FR and T_{exec} indicate the contributions of each stage to the fix rate and execution time, respectively.
[2] The column labeled UVLLM summarizes the total contributions across all stages of the repair operation.
[3] Modules are grouped as Arithmetic (Accumulator, Adder, Divider, Multiplier), Control (Counter, FSM), Memory, and Miscellaneous (other modules).
[4] Errors are categorized as syntax ("s") and function ("f").

Table II details the repair process, which unfolds in several stages, each playing a different role in resolving syntax and functional errors. The Pre-processing stage was particularly effective in addressing syntax errors, successfully resolving 74.72% of these cases as highlighted. For functional errors, the Mismatch Signal (MS) mode in the Repair stage was most effective, correcting 41.46% of the instances. Notably, while most syntax errors were rectified during pre-processing, 11.29% were forwarded to the MS mode. Moreover, 25.96% of functional error repairs inadvertently introduced syntax errors, which were subsequently managed by the pre-processor. This demonstrates UVLLM's capacity to compensate for errors introduced in earlier stages and mitigate uncertainties inherent in LLMs.

In terms of execution time, the segmented repair operation shows that the pre-processing stage, despite handling over 50% of all benchmark repairs, typically requires less time than repairs conducted in MS mode. This demonstrates the efficiency benefits of incorporating a robust pre-processing stage.

Result 5: The UVLLM framework's execution efficiency was evaluated against existing methods from a multidimensional perspective, mainly focusing on two key metrics: Failure Rates (FRs) and execution time T_{exec}, as detailed in Table II.

While Fig.5 and Fig.6 show the variations in FR performance across different error types, UVLLM consistently surpassed MEIC across all module types. For instance, within the *Miscellaneous* modules for syntax errors in Table II, UVLLM achieved an FR of 88.50%, which is 21.83% higher than MEIC's 66.67%. In handling functional errors within *Arithmetic* modules, characterized by their complex logic, UVLLM maintained an FR of 66.23%, significantly outperforming MEIC's 40.53%. These results underscore UVLLM's robust capability to effectively resolve a wide spectrum of errors across different modules compared to existing methods, thus proving its effectiveness and reliability in boosting system performance.

In terms of operational efficiency, UVLLM also demonstrated a substantial reduction in execution time. For example, when processing syntax errors within the *Miscellaneous* modules, UVLLM recorded an average execution time of 13.47s, marking a speedup of 4.65x compared with MEIC. This advantage was even more pronounced when addressing complex functional errors, where UVLLM achieved a speedup of up to 16.56x over MEIC in *Arithmetic* modules. On average, UVLLM operated 10.42x faster than MEIC, while simultaneously achieving higher test coverage and pass rates. These findings highlight UVLLM's potential to significantly enhance the design verification process for practical deployment by merging increased automation with superior efficiency.

C. Ablation Study

Our research includes the ablation study designed to evaluate the impact of iteration strategies on the effectiveness of the framework.

TABLE III: Ablation study. UVLLM$_{pair}$ and UVLLM$_{comp}$ represent UVLLM with LLMs generating code pairs and complete codes.

Framework	$FR/\%$		T_{exec}/s	
	Syntax	Func.	Syntax	Func.
$UVLLM_{pair}$	86.99	71.92	13.83	15.79
$UVLLM_{comp}$	70.41	59.25	35.60	71.84

Repair generation form. We initially employ an approach that uses original-repair code pairs to facilitate the generation of new code by leveraging outputs from LLMs. However, the ablation study examines an alternative method where entire code snippets are directly produced by the LLMs, omitting the generation of repair pairs.

As shown in Table III, generating complete code snippets resulted in a slight decline in repair accuracy and an increase in execution time compared to generating original-repair code pairs. Nevertheless, there were specific scenarios where this direct generation method proved superior. This advantage is mainly due to the ability of the direct generation method to handle minor errors that pose significant challenges for LLMs in terms of search efficiency. In some cases, regenerating the entire code is more effective than trying to modify or replace segments of the existing code. For instance, correcting the error "$module\ a(A); ...(Missing\ Definition\ of\ Port\ A)...endmodule$" proves challenging for the replacement strategy, primarily because the essential context is frequently overlooked, whereas the reproduction method handles it more straightforwardly.

V. Conclusion

In this work, an automated universal verification framework, **UVLLM**, which comprehensively addresses main phases of hardware design verification, including testbench construction, test execution, result analysis, and repair, is proposed. Tested on the proposed benchmark, UVLLM achieves average syntax and functional error fault rates of 86.99% and 71.92%, respectively, while maintaining nearly 100% test coverage. Additionally, UVLLM performs 10.42 times faster than the previous MEIC framework. The framework demonstrates that it is feasible to employ the LLMs for the purpose of Verilog code verification, irrespective of the initial code state. The utilization of reasonable segmentation and feedback engineering leads to an improvement in the verification efficiency of the framework.

VI. Acknowledgement

This work is supported by the National Key Research and Development Program (No. 2024YFB4405600), the National Natural Science Foundation of China (No. 62472086), the Basic Research Program of Jiangsu (No. BK20243042), the Key R&D Program of Jiangsu Province (No. BE2023020-1), and the Science and Technology Major Special Program of Jiangsu (No. BG2024010).

REFERENCES

[1] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? a study on the state of high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, 2018.

[2] Y. Liu, L. Zhang, and Z. Zhang, "A survey of test based automatic program repair." *J. Softw.*, vol. 13, no. 8, pp. 437–452, 2018.

[3] K. Liu, L. Li, A. Koyuncu, D. Kim, Z. Liu, J. Klein, and T. F. Bissyandé, "A critical review on the evaluation of automated program repair systems," *Journal of Systems and Software*, vol. 171, p. 110817, 2021.

[4] Q. Zhang, T. Zhang, J. Zhai, C. Fang, B. Yu, W. Sun, and Z. Chen, "A critical review of large language model on software engineering: An example from chatgpt and automated program repair," *arXiv preprint arXiv:2310.08879*, 2023.

[5] X. Yin, C. Ni, S. Wang, Z. Li, L. Zeng, and X. Yang, "Thinkrepair: Self-directed automated program repair," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 1274–1286.

[6] K. Xu, G. L. Zhang, X. Yin, C. Zhuo, U. Schlichtmann, and B. Li, "Automated c/c++ program repair for high-level synthesis via large language models," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024, pp. 1–9.

[7] H. Ahmad, Y. Huang, and W. Weimer, "Cirfix: automatically repairing defects in hardware design code," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 990–1003.

[8] D. Yang, J. He, X. Mao, T. Li, Y. Lei, X. Yi, and J. Wu, "Strider: Signal value transition-guided defect repair for hdl programming assignments," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 5, pp. 1594–1607, 2024.

[9] K. Laeufer, B. Fajardo, A. Ahuja, V. Iyer, B. Nikolić, and K. Sen, "Rtl-repair: Fast symbolic repair of hardware design code," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 867–881.

[10] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–8.

[11] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *arXiv preprint arXiv:2308.00708*, 2023.

[12] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," *arXiv preprint arXiv:2305.13243*, 2023.

[13] M. DeLorenzo, A. B. Chowdhury, V. Gohil, S. Thakur, R. Karri, S. Garg, and J. Rajendran, "Make every move count: Llm-based high-quality rtl code generation using mcts," *arXiv preprint arXiv:2402.03289*, 2024.

[14] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu *et al.*, "Chipnemo: Domain-adapted llms for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[15] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language models," *arXiv preprint arXiv:2311.16543*, 2023.

[16] X. Yao, H. Li, T. H. Chan, W. Xiao, M. Yuan, Y. Huang, L. Chen, and B. Yu, "Hdldebugger: Streamlining hdl debugging with large language models," *arXiv preprint arXiv:2403.11671*, 2024.

[17] K. Xu, J. Sun, Y. Hu, X. Fang, W. Shan, X. Wang, and Z. Jiang, "Meic: Re-thinking rtl debug automation using llms," *arXiv preprint arXiv:2405.06840*, 2024.

[18] W. Fu, K. Yang, R. G. Dutta, X. Guo, and G. Qu, "Llm4sechw: Leveraging domain-specific large language model for hardware debugging," in *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2023, pp. 1–6.

[19] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382*, 2023.

[20] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A systematic survey of prompt engineering in large language models: Techniques and applications," *arXiv preprint arXiv:2402.07927*, 2024.

[21] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[22] OpenAI, "GPT-4 pricing for OpenAI API," 2024. [Online]. Available: https://openai.com/api/pricing/

[23] F. Plasencia-Balabarca, E. Mitacc-Meza, M. Raffo-Jara, and C. Silva-Cárdenas, "Robust functional verification framework based in uvm applied to an aes encryption module," in *2018 New Generation of CAS (NGCAS)*. IEEE, 2018, pp. 194–197.

[24] C. Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. Springer, 2008, vol. 161.

[25] L. Séméria and A. Ghosh, "Methodology for hardware/software co-verification in c/c++ (short paper)," in *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*, 2000, pp. 405–408.

[26] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, and T. Yoshimura, "A fast hardware/software co-verification method for system-on-a-chip by using a c/c++ simulator and fpga emulator with shared register communication," in *Proceedings of the 41st annual Design Automation Conference*, 2004, pp. 299–304.

[27] V. Radu, D. Dranga, C. Dumitrescu, A. I. Tabirca, and M. C. Stefan, "Generative ai assertions in uvm-based system verilog functional verification," *Systems*, vol. 12, no. 10, p. 390, 2024.

[28] Z. Ji, T. Yu, Y. Xu, N. Lee, E. Ishii, and P. Fung, "Towards mitigating llm hallucination via self reflection," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 1827–1843.

[29] X. Amatriain, "Measuring and mitigating hallucinations in large language models: amultifaceted approach," 2024.

[30] B. A. Galitsky, "Truth-o-meter: Collaborating with llm in fighting its hallucinations," 2023.

[31] Y. Chen, Q. Fu, Y. Yuan, Z. Wen, G. Fan, D. Liu, D. Zhang, Z. Li, and Y. Xiao, "Hallucination detection: Robustly discerning reliable answers in large language models," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 245–255.

[32] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[33] N. Lambert, V. Pyatkin, J. Morrison, L. Miranda, B. Y. Lin, K. Chandu, N. Dziri, S. Kumar, T. Zick, Y. Choi *et al.*, "Rewardbench: Evaluating reward models for language modeling," *arXiv preprint arXiv:2403.13787*, 2024.

[34] OpenAI, "Structured outputs - OpenAI API," 2024. [Online]. Available: https://platform.openai.com/docs/guides/structured-outputs

[35] S. Sudakrishnan, J. Madhavan, E. J. Whitehead Jr, and J. Renau, "Understanding bug fix patterns in verilog," in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 39–42.

[36] J. Ma, G. Zuo, K. Loughlin, H. Zhang, A. Quinn, and B. Kasikci, "Debugging in the brave new world of reconfigurable hardware," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 946–962.

[37] V. Antinyan, M. Staron, and A. Sandberg, "Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time," *Empirical Software Engineering*, vol. 22, pp. 3057–3087, 2017.

[38] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," *arXiv preprint arXiv:2308.05345*, 2023.

[39] Synopsys, "VCS: Synopsys Verification Continuum," 2024. [Online]. Available: https://www.synopsys.com/verification/simulation/vcs.html

[40] I. V. Team, "Icarus Verilog," 2024. [Online]. Available: http://iverilog.icarus.com/

[41] Siemens, "ModelSim," 2024. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/modelsim/

[42] C. Wolf, "Yosys Open SYnthesis Suite," 2024. [Online]. Available: http://www.clifford.at/yosys/

[43] W. Snyder, "Verilator," 2024. [Online]. Available: https://www.veripool.org/wiki/verilator

[44] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[45] S. Fakhoury, A. Naik, G. Sakkas, S. Chakraborty, and S. K. Lahiri, "Llm-based test-driven interactive code generation: User study and empirical evaluation," 2024.

[46] R. Tian, Y. Ye, Y. Qin, X. Cong, Y. Lin, Z. Liu, and M. Sun, "Debugbench: Evaluating debugging capability of large language models," *arXiv preprint arXiv:2401.04621*, 2024.

A Full-system, Programmable, and Extensible In-Memory Computing Simulation Framework for Deep Learning

Kaining Zhou, Jian Huang, Nam Sung Kim, Naresh Shanbhag
University of Illinois at Urbana-Champaign, IL, USA
{kainingz, jianh, nskim, shanbhag}@illinois.edu

***Abstract*—In-memory computing (IMC) has established itself as an attractive alternative to hardware accelerators in addressing the memory wall problem for artificial intelligence (AI) workloads. However, designing programmable IMC-based computing platforms for today's large generative AI models, such as large language models (LLMs) and diffusion transformers (DiTs), is hindered by the absence of a simulator that is able to address the associated scalability challenges while simultaneously incorporating device and circuit-level behaviors intrinsic to IMCs. To address this challenge, we present *IMCsim*, a versatile full-system IMC simulation framework. IMCsim integrates software runtime libraries for AI models, introduces a new set of ISA extensions to express common tensor operators, and provides flexibility in mapping these operators to various IMC architectures. As such, IMCsim enables designers to explore trade-offs between performance, energy, area, and computational accuracy for various IMC design choices. To demonstrate the functionality, efficiency, and versatility of IMCsim, we model three types of IMCs: (1) embedded non-volatile memory (eNVM)-based, (2) SRAM-based, and (3) digital IMCs. We validate IMCsim using measured data from two laboratory-tested IMC prototype ICs—a $22\,\mathrm{nm}$ MRAM-based IMC and a $28\,\mathrm{nm}$ SRAM-based IMC—and a digital IMC design in $28\,\mathrm{nm}$. Next, we demonstrate the utility of IMCsim by exploring the architectural design space to obtain insights for maximizing utilization of IMC-based processors for diverse workloads—ResNet-18, Llama, and a DiT—using the three IMC types. Finally, we employ IMCsim as a design tool to obtain an efficient chip architecture and layout in $28\,\mathrm{nm}$ for a lightweight DiT.**

I. Introduction

In-memory computing (IMC) [17] is a promising approach that incorporates computation into memory arrays, thereby alleviating the memory bandwidth bottleneck of conventional von Neumann architectures. By enabling data-intensive tasks such as matrix-vector multiplication (MVM) within memory, IMC is well-suited for AI workloads built upon deep nets [12] and transformers [28], exemplified by a collection of IC prototypes [6], [7], [11], [15], [16], [26], [27], [29], [31].

The design of an IMC-based system is complex. Designers innovate across the compute stack: device, circuit, architecture, and workload mapping. Therefore, there is a critical need for a fast IMC simulator to enable the design and validation of full IMC-based systems. Such a simulator should provide full-system, cycle-level run-time behavior, with support for instruction set architecture (ISA), while capturing the impact of device and circuit non-idealities.

Previously developed IMC simulators (see Table I) fall into one or more of the following three classes: 1) **resistive crossbar macro simulators**: these characterize circuit-level metrics of IMC crossbar macros [14], [19], [33]; 2) **architecture behavioral simulators**: these simulate behavior of IMC architectures when executing applications [2], [32]; and 3) **design space exploration tools**: these enable fast exploration of efficient IMC designs using statistics-based modeling and predefined mapping [1], [33]. In most cases, these existing simulators can handle only small-scale workloads without considering the impact on an end-to-end IMC system, and therefore are unable to serve as a design tool for an actual chip implementation.

To this end, we develop IMCsim, a versatile full-system IMC simulation framework for designing IMC-based systems capable of executing large AI models such as large language models (LLMs). Unlike standalone static device/circuit characterization tools, IMCsim integrates off-chip memory access and run-time environment into the simulator. To support simulation of various types of workloads on diverse architectures, IMCsim provides flexible programmable interfaces (see Section II-B). Further, IMCsim allows designers to decompose deep learning (DL) kernels into basic tensor operators and proposes a set of extensible ISA that translates these operators into the instructions that can be executed on IMC processors. Furthermore, instead of focusing on a single type of memory device, IMCsim can support different memory technologies such as SRAM, ReRAM, and MRAM.

IMCsim enables cycle-level behavioral simulation by providing accurate timing for each circuit operation. It also employs well-acknowledged power models for various memory devices and IMC processors, for precise power estimation. To guarantee IMC accuracy, IMCsim develops a signal-to-noise ratio (SNR) model that quantifies the computational accuracy.

We implement IMCsim and integrate it into a full-system emulator, QEMU [5]. To evaluate the efficacy of IMCsim, we use three types of DL applications, including ResNet-18 [12], Llama [25], and Diffusion Transformer (DiT) [21] on SRAM-based and MRAM-based analog IMC (AIMC), as well as SRAM-based digital IMC (DIMC). We show the end-to-end performance, energy consumption, and computation accuracy from the simulation, validated by fabricated IC and complete layouts. As IMCsim reports detailed run-time information on the status, utilization, and energy of each component, it will benefit future IMC development by offering useful architectural insights. Overall, the contributions of this work include:

1) **Full-system, programmable, and extensible simulator**: we present IMCsim, a versatile simulator allowing researchers to utilize hardware libraries to model diverse IMC processors with ISA support atop various IMC devices. IMCsim is calibrated with data from fabricated chips and chip layouts. Its built-in expressive application programming interfaces facilitate designers to program and map workloads correspondingly. IMCsim is open-sourced at https://anonymous.4open.science/r/IMCsim-2B45/.
2) **Accurate and insightful evaluation for large-scale AI models**: we employ IMCsim's cycle-level run-time information, noise analysis, and interaction with external memory systems, to identify bottlenecks and highlight optimization opportunities when mapping Llama and DiT to three different IMC architectures, respectively.
3) **Design exploration targeting chip implementation**: we demonstrate the use of IMCsim to conduct a design space exploration for implementing an IMC chip layout for a lightweight DiT in $28\,\mathrm{nm}$ process technology. This exploration involves architectural and functional design, evaluation, verification, and optimization of the IMC system illustrating IMCsim's utility as a design tool.

We organize the rest of the paper as follows: Section II highlights

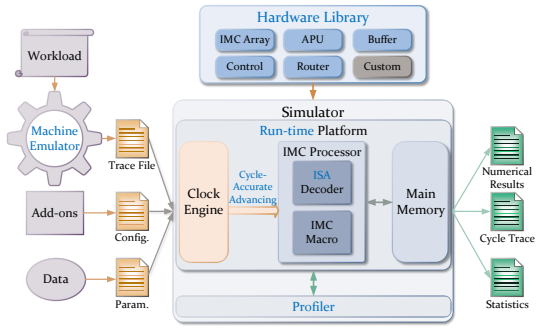


Fig. 1: System architecture of IMCsim.

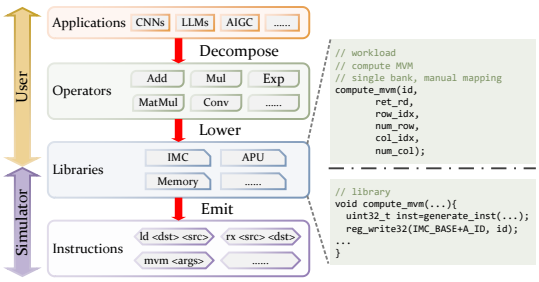


Fig. 2: Software stack and programming interface of IMCsim.

design features of IMCsim, Section III shows how validation is performed, and Section IV demonstrates multiple use cases to show IMCsim's efficacy, and Section V concludes the paper.

II. IMCsim Features

This section describes key IMCsim features including circuit and architecture model, cycle-level behavior, extensible ISA, run-time profiling, and others.

A. Hardware Library

IMCsim provides a hardware library comprising several "building blocks" (see Fig. 1) to enable designers to compose a diversity of IMC-based architectures.

Basic blocks: IMCsim has five common blocks found in current day IMC processors: an IMC array, an auxiliary processing unit (APU), a buffer, a controller, and a router. The IMC array is the core memory array for MVM computation with the following parameters: the number of rows and columns, the number of ADCs and adder trees, the I/O width, and their timing information. The APU is a digital block comprising several types of computational units, including fixed-point and floating-point ALUs. It can be organized and executed in a vectorized manner. The combined effort of the IMC array and the APU makes IMC capable of processing general AI workloads. The controller executes instruction fetch, and decode, for the IMC bank. The buffer is an on-chip SRAM that stores intermediate results and buffers data transmitted through the I/O. The router connects multiple IMC banks.

Custom blocks: Designers have the flexibility to incorporate their own components to this simulator, *e.g.*, sparsity detection units, and quantization units. Section II-F provides more details.

B. Programmability

Compared with related IMC modeling tools [14], [19], IMCsim provides programmability across diverse applications, architectures and devices (see Fig. 2). Users select an application and express it using common operators *e.g.*, those found in ONNX [3]. Next,

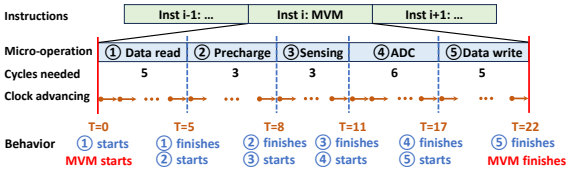


Fig. 3: Behavioral cycle-level modeling for MVM instruction.

software operators are mapped onto hardware-supported operations via library calls such as `compute_mvm`. Such flexibility allows users to run various applications and evaluate different mappings of applications on different architectures. Thus, IMCsim is not confined to a predefined set of workloads or architectures, making it highly versatile.

C. ISA Support

Compared to related tools [14], [32], IMCsim provides an additional level of flexibility via a specialized and extensible RISC style ISA to designers as they compose their IMC processor. Table II lists the most frequently used instructions obtained by analyzing DL workloads and IMC designs. Instructions in Fig. 2 adhere exactly to this ISA. The ISA allows for easy implementation and easily extensible to a specific accelerator. The trace file (Fig. 1) is formatted to drive IMCsim.

D. Cycle-level Simulation

IMCsim overcomes the limitations of existing tools in simulating complex workloads at system and application levels by offering cycle-level behavioral simulation, and balancing the abstraction between circuit and application while specifying operation timing. Applications are decomposed into detailed operators by the user, and translated into hardware instructions (last level in Fig. 2). Meanwhile, lower-level circuits are abstracted into building blocks (IMC arrays, APUs, buffers) ensuring a comprehensive simulation for system and application-level designs. IMCsim models timing by decomposing one instruction into several micro-operations, each of which is associated with a hardware component with an execution time. This association is determined by the user.

For instance, to process one MVM, the system needs to execute 5 micro-operations which the user derives from real circuit behavior and provides to the simulator. As shown in Fig. 3, at T=0, IMCsim begins to execute the micro-operation ① which is data read. The clock engine increments the time T until T=5, which is the cycle count for ①, denoting its completion. At T=5, the input vector for MVM is updated thereby simulating data read from on-chip memory. The simulation proceeds to simulate the precharge micro-operation ② next. Only after the data write micro-operation ⑤ is executed, is the MVM instruction completed and committed.

E. Run-time Support

IMCsim provides run-time support since it is necessary to expose errors to the user during execution. It also enables the user to identify bottlenecks and optimization opportunities unavailable at compile-time. We list IMCsim's run-time support as follows: (1) determine run-time behavior per cycle per component, *e.g.*, the specific bank executing a specific instruction and buffer addresses being accessed; (2) perform numerical computations to monitor intermediate data easily; (3) monitor component behavior and extract metrics such as energy and utilization, *i.e.*, it dynamically profiles metrics instead of predicting them statically at compile-time.

TABLE I: Landscape of various IMC simulators

Work	Feature	Type of memory	Application capability	Cycle-level	Run-time support	ISA support	Noise analysis	Full-system simulation
PUMAsim [2]	PUMA-specific architecture simulator	Memristor	ML, CNNs	Yes	Yes	Yes	No	No
RxNN [14]	Fast crossbar simulation with non-ideality	ReRAM	CNNs	No	No	No	Yes	No
MNSIM [33]	CNN-specific IMC design framework	Memristor	CNNs	No	No	No	Yes	No
NeuroSim [19]	Device-accurate crossbar simulator	NVM	MLPs, CNNs	No	No	No	Yes	No
PIMulator-NN [32]	Event-driven crossbar simulator	NVM, SRAM	MLPs, CNNs	No	Unknown	No	No	No
CiMLoop [1]	Fast and flexible IMC modeling tool	User-defined	CNNs, Transformers	No	No	No	No	Yes
IMCsim (our work)	**Full IMC system design tool**	User-defined	General AI workloads	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

TABLE II: A snapshot of base ISA

31 25	24 20	19 15	14 12	11 7	6 0	
000000	X	rs1	000	rd	0000000	LOAD
000001	rs2	rs1	000	X	0000000	STORE
000010	rs2	rs1	000	rd	0000000	TX
000000	X	X	000	rd	0000001	MVM
000010	rs2	rs1	000	rd	0000001	ADD
000100	rs2	rs1	001	rd	0000001	FADD

F. Extensiblity

IMCsim preserves extensibility in two respects:

ISA extension: IMCsim's set of basic instructions (Table II) can be easily extended. For example, the ISA can be extended to support BF16 (16-bit brain floating-point) data arithmetic by: (1) specifying the format of the new instruction; (2) adding the corresponding functions in the software library; and (3) modify the tracing function in the machine emulator (Fig. 1) to correctly detect the execution of the instruction for collecting the traces. The ISA extension needs to be accompanied by an extension to the hardware as discussed below.

Hardware extension: IMCsim enables the user to include custom blocks to extend their hardware by: (1) creating a class, *e.g.*, class of BF16 hardware under the class of IMC macro, and elaborate its definition and function in the simulator; (2) profile the hardware in advance to characterize its metrics such as latency, energy, and area; and (3) define its behavior in each cycle as the clock is advanced; (4) add a function to advance its clock. Thus, the function of BF16 hardware will be invoked whenever the new BF16 instruction is executed.

G. SNR Estimation

IMCsim constructs an SNR model to address fixed-point digital-based and representative analog-based IMCs. Compute SNR is defined by the ratio of variance of the ideal output and variance of the deviation from the ideal (noise) output. Specifically, the compute SNR for a dot product (DP) of $\mathbf{w}$ and $\mathbf{x}$, ideal output y_o, the actual output y, noise η, then compute SNR is given as follows.

$$y_o = \mathbf{w}^{\mathbf{T}}\mathbf{x} = \sum_{j=1}^{N} w_j x_j \tag{1}$$

$$y = y_o + \eta \tag{2}$$

$$\text{Compute SNR} = 10\log_{10}\left(\frac{\sigma^2[y_o]}{\sigma^2[\eta]}\right)(\text{dB}) \tag{3}$$

where y_o is the ideal output, y is the actual output, and η is the deviation or noise. IMCsim takes the configuration and actual data as input, including the dimension of IMC arrays, bitwidth of ADCs (or output precision), inputs and weights, their bitwidth, and other parameters to compute y and then employs (3) to estimate the compute SNR. For AIMCs, the impact of circuit parameters such as thermal noise, deviation of capacitance, and supply voltage are incorporated through pre-characterization of the hardware components.

H. Full-system Integration

Main memory: IMCsim accounts for traffic to/from main memory in its simulation set-up. In the process, it reports the latency and energy of such off-chip memory accesses which can dominate the system latency and energy consumption. In the process, IMCsim measures the impact of accessing main memory by an IMC processor thereby providing a more complete assessment of the system to the user.

Full-system emulator: IMCsim is integrated into the full-blown generic machine emulator QEMU (Fig. 1) to further enhance its programmability and completeness as a design tool. IMC macros can be added to a machine as a peripheral device through MMIO or a PCI device. The user can define IMC macro specifications and program it in C to process DL algorithms by applying particular mapping schemes. After invoking QEMU, the transaction between the CPU and IMC device is captured automatically after which IMCsim begins its simulation.

III. Simulator Validation

We use three chips for validation: (1) measured data from a $22\,\text{nm}$ MRAM-based AIMC chip to calibrate one exemplary NVM-based IMC device; (2) measured data from a $28\,\text{nm}$ SRAM-based AIMC chip to be our representative volatile memory-based IMC device; (3) and simulated data from a $28\,\text{nm}$ DIMC chip.

Timing validation: IMCsim supports in-order, single-stage execution in the IMC processor for simplicity and ease of verification. We profile the hardware components to obtain their execution times for each micro-operation and provide it as inputs to IMCsim so that it can provide cycle-level timing behavior for the system. For example, for timing characterization of IMC banks, we obtain the latency of each micro-operation, such as data read/write from circuit simulations or chip measurements, to obtain the cycle-level timing for an MVM instruction as shown in Fig. 3. For digital components such as controller, fixed- and floating-point digital blocks, we simulate their behavioral RTL models for functional verification and synthesize them to obtain their latency. Lastly, we simulate the IMC system in IMCsim and compare the resulting system-level outputs with those obtained from software (Pytorch and C) as behavioral ground truth. This validates both the behavior and timing of the IMC system.

IMCsim allows for new designs and instruction extensions to be incorporated. To do this, the user can profile the unit cycle needed by new components in advance and include it to a cycle list maintained internal to the simulator.

Metric validation: For metrics besides latency, we characterize hardware in advance at the circuit level as well. We measure our IMC chips to obtain energy consumption for each stage and area overhead breakdown for the IMC array. In addition, we generate layouts and conduct post-extraction simulations to obtain the energy and area numbers. Further flexibility and extensibility are provided by add-ons in IMCsim, including [4] for SRAM and DRAM metrics, and [19] for SRAM and NVM metrics. Moreover, a technology scaling tool [23] is used, enabling users to explore the impact of technology scaling.

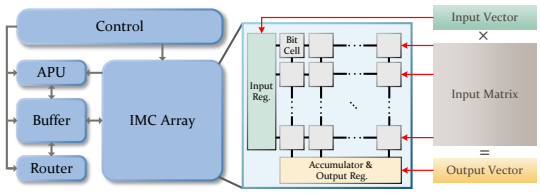


Fig. 4: IMCsim's flexible architectural template for composing IMC systems.

TABLE III: Design configurations employed in the case studies.

Component	Specification	Latency	Energy (fJ)	Area (mm^2)
DRAM	128b IO	25.8 ns/access	5.35×10^6	-
Buffer	192KiB, 128b IO	1.6 ns/access	29×10^3	1.00
Controllor	-	-	6×10^3/cycle	2.036×10^{-3}
Router	16b IO	16 B/2cycles	11	8.835×10^{-3}
APU	2 copies of ALUs	1-30 cycles	116-1477	0.19

IMC Array	Operation	Cycles	Energy (fJ)	Area (mm^2)
SRAM AIMC 512×512	Row read	4	1.02×10^5	0.19
	Row write	4	2.30×10^5	
	1b input driver	1	1.5	
	8b ADC	16	5.2×10^3	
MRAM AIMC 512×512	Row read	4	1.20×10^5	0.86
	Row write	50	4.6×10^7	
	1b sensing	2	98	
	8b ADC	8	3.1×10^3	
SRAM DIMC 512×512	Row write	4	2.30×10^5	1.64
	MAC	1	1.5/1b-op	

Functionality and SNR Validation: We verified functionality with software generated ground truth [9], [18], [20]. For volatile memory, we calibrated different types of SNR calculation with our internal SRAM-based IMC chip and Cadence simulations. For NVM-based IMC, we verify its SNR with our MRAM chip prototype, which is representative of the popular IMC of the NVM type. The average error in our SNR model is around 5% compared to chip measurements.

IV. Design Case Studies

In this section, we present case studies to highlight the use of IMCsim in designing IMC-based systems by demonstrating its ability to provide insights into performance, power, and area, of multiple designs as well as its ability to aid design verification of the prototype IC. We assume the generic architectural template in Fig. 4 is employed by the designer. The design configurations in Table III are employed for generating IMC designs. Three IMC types are considered: 1) SRAM AIMC; 2) MRAM AIMC; and 3) digital IMC (DIMC). Data for the IMC configurations were obtained from lab-tested/tape-out chip prototypes described in Section III.

We select three diverse workloads: 1) ResNet-18 [12] (11M parameters) on ImageNet [8] for the image classification task, representing CNN-type workloads; 2) Llama 6.7B [25] for the text generation task, representing transformer decoder-only autoregression workload; and 3) diffusion transformer [21], DiT-XL/2 (675M parameters) for the image generation task, representing transformer-based diffusion workloads [13]. For ResNet-18, the entire network is simulated and verified. However, without loss of generality, we simulate only one transformer layer of Llama and DiT, since the transformer model is built by repeating this layer.

A. Basic Profiling

Performance across IMC types: We employ the roofline plot [30] (in Fig. 5) to demonstrate how the three AI models perform on the three IMC processors, which are identical except for the IMC type.

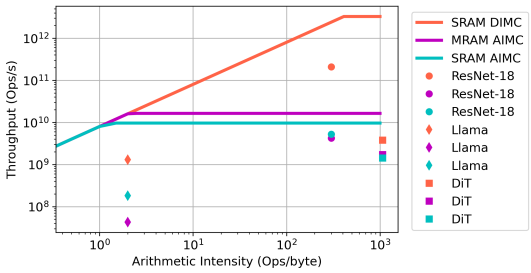


Fig. 5: The roofline plots generated by IMCsim.

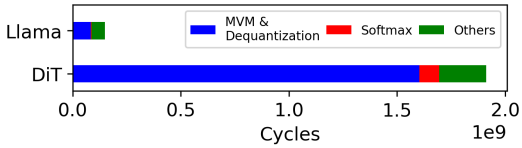


Fig. 6: Cycle-level breakdown for DIMC for sequence length = 256.

We find, DIMC achieves the highest peak throughput across all three workloads, which is consistent IMC benchmarking data [24]. This is due to DIMC's fast MAC operations, e.g., single-cycle MAC at 500 MHz) as compared to AIMC architectures, which rely on low ADC sampling rates, e.g., 20 MHz, to execute an MVM.

Performance across model types: ResNet-18 achieves the highest throughput due to high reuse enabled by its convolutional kernels and less frequent weight reloading. In Llama and DiT, MVM and dequantization dominate computation time when the sequence length is small, e.g., 256, which is consistent with the analysis in [22] as seen in Fig. 6. As a decoder-only transformer with a KV-cache, Llama performs MVM and softmax only on newly generated tokens. DiT, however, processes all tokens at each step, making its MVM and softmax operations much more computationally demanding.

B. Utilization Analysis

Utilization of IMC is measured by the number of memory rows and columns activated during MVM operations. Using DIMC as an example, Fig. 7 shows ResNet-18's utilization across layers. The first layer has the lowest utilization and longest run-time, while later layers sometimes reach 100% utilization due to larger kernels mapped to the IMC array. Periods of 0 utilization indicate idle IMC banks, possibly due to weight loading or waiting for other processes.

Utilization of transformer models is impacted by quantization group size. ResNet-18 uses per-layer quantization, but large models such as Llama and DiT require finer-grained quantization [10]. With group sizes of 64 for Llama and 128 for DiT, utilization fluctuates between 2 values - 0 and 12.5% or 25% - respectively, showing significant underutilization of the 512-row IMC bank and hence a lower throughput.

C. Few Large Cores vs. Many Small Cores

The choice between a few powerful cores and many small cores is critical in IMC design. Earlier AIMC designs [15] favored large cores, e.g., up to 1152 rows, but recent studies [26] have reduced core sizes to as few as 16 rows in DIMC. To investigate core size and number, we varied DIMC configurations for large models.

Table IV shows that splitting banks into smaller ones (*e.g.*, two 512×256 banks) offers no benefit, as columns are fully utilized in Llama and DiT. Scaling should instead focus on the row dimension. For example, though Llama is identified as memory-bound in the roofline plot (Fig. 5), its performance on DIMC can be improved

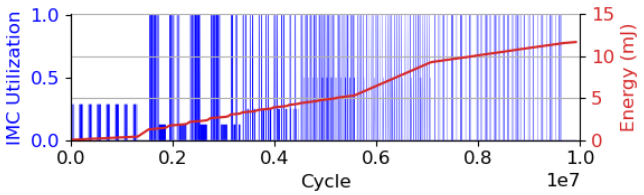


Fig. 7: Utilization and energy while executing ResNet-18 obtained via IMCsim's run-time profiling.

TABLE IV: Scaling results for large models obtained by IMCsim

Workload	DIMC bank specification	Number of banks	Relative peak throughput	Actual throughput	Average utilization (%)
Llama	512 × 256	2	1×	1.0×	0.09
	256 × 512	2	1×	2.0×	0.19
	128 × 512	2	0.5×	2.0×	0.19
	64 × 512	2	0.25×	2.0×	0.19
	64 × 512	4	0.5×	3.9×	0.36
DiT	512 × 256	2	1×	1.0×	1.9
	256 × 512	2	1×	2.0×	3.9
	128 × 512	2	0.5×	2.0×	3.9
	128 × 512	4	1×	3.9×	7.8
	128 × 512	8	2×	8.1×	15.4

by 2× by the use of improved bank dimensions (128 × 512) and more banks (2), which leads to enhanced utilization. Therefore, a key insight is that reducing IMC capacity can enhance throughput, especially when the number of IMC rows matches the quantization group size, thereby maximizing utilization. Another noteworthy observation is that the throughput enhancement in a multi-bank IMC can be slightly more than linear in the number of banks (8-bank IMC's throughput is 8.1× that of a single bank IMC) since multi-bank computations effectively mask idle cycles which single-bank IMCs are unable to.

These findings indicate multiple small cores are better suited for large models, paralleling trends in CPU design, where finer-grained scheduling and higher utilization are key for efficient hardware.

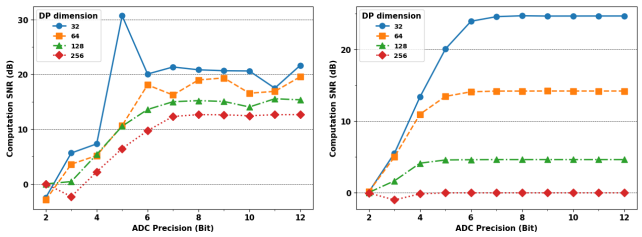
D. Compute SNR Analysis

Fig. 8 shows the impact of ADC bit precision on the computational SNR. It is generated by computing the DP of random input data. One observes that SRAM-based IMCs achieve higher SNR than MRAM-based ones. This is consistent with observed data [24]. The compute SNR for SRAM shows a peak when the ADC precision equals $\log_2(N)$, where N is the DP dimension. This peak occurs because each DP output has a unique ADC quantization level associated with it. Finally, the compute SNR increases as the N reduces, because the dynamic range of the ADC input is reduced. Such insights can guide circuit designers to choose ADC precision and dot product dimensions associated with specific quantization group sizes.

E. Energy Analysis

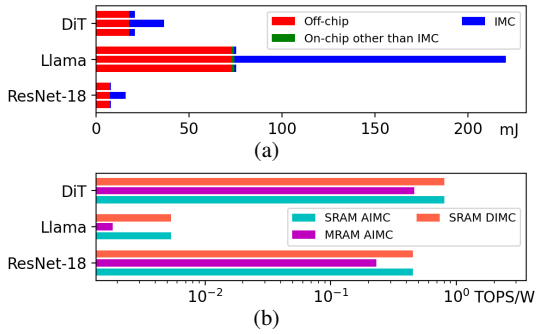
Fig. 9 profiles the system energy consumption and distribution for three AI workloads mapped to three IMC types. As expected, off-chip access dominates in all cases except MRAM-based IMC. The reason for this discrepancy is the high write energy of MRAM compared to SRAM and DIMC (see Table III). Furthermore, Llama requires more writes than DiT and ResNet-18 due to its much larger parameter count (6.7B vs. 675M for DiT and 11M for ResNet-18).

Interestingly, the energy efficiency of all applications is below 1 TOPS/W, which is about 10× lower than those reported in typical IC prototypes [24]. We conjecture this discrepancy results from accounting for the energy cost of frequent off-chip accesses which is typically not reported.



(a) (b)

Fig. 8: SNR for: (a) SRAM-, and (b) MRAM-based AIMCs.



(a)

(b)

Fig. 9: System energy profiling of three workloads mapped onto three IMC types (from top to bottom: DIMC, MRAM-based, and SRAM-based): (a) system energy distribution, and (b) system energy efficiency.

F. IMCsim as a Design Tool

This section illustrates the use of IMCsim as a design tool in implementing an prototype IMC chip.

Application specification: Consider the implementation of fixed-point MVMs kernels of one layer of a lightweight DiT algorithm on an IMC chip tailored for Edge having the following algorithmic parameters: *sequence length* = 256; *fixed-point precision* = 8-bit; *quantization group size* = 32 (attention) or 128 (otherwise); *MLP ratio* = 4. We assume non-MVM operations such as dequantization and non-linearity are executed off-chip.

Hardware specification: Assume an area budget of $4\,\mathrm{mm}^2$ in a $28\,\mathrm{nm}$ process. Based on the analyses in Section IV-B, the architecture needs to support a DP dimension consistent with the quantization group size (=32) to maximize utilization. Therefore, we choose a DIMC bank featuring 32 × 32 SRAM cells. The configuration file (see Fig. 1) will include the following parameters: chip area $A_{chip} = 4\,\mathrm{mm}^2$, percentage of area available for IMC and SRAM banks $P = 50\%$ (to leave room for the controller, pad ring and other support circuitry), standard SRAM bitcell with area $A_{bc} = 0.127\,\mu\mathrm{m}^2$; IMC bank unit area $A_{bank} = 16000\,\mu\mathrm{m}^2$; frequency $f = 500\,\mathrm{MHz}$; off-chip I/O bitwidth $B = 32$.

If the number of memory cells is M and the number of IMC banks is N_{bank}, then we have the following constraint:

$$\frac{A_{bc}}{F} M + A_{bank} N_{bank} \leq P A_{chip} \tag{4}$$

where F is the fraction of the total area of the SRAM banks that is occupied by bitcells.

Baseline Architecture: For the baseline case ($N_{bank} = 1$), with a typical value $F = 0.65$, (4) suggests a maximum on-chip memory capacity $M_{max} \leq 10^7$ (10 Mb). Employing the architectural template in Fig. 4 with one IMC 32 × 32 bank and 20 4096 × 128 SRAM banks, we employ IMCsim to profile the memory footprint of one

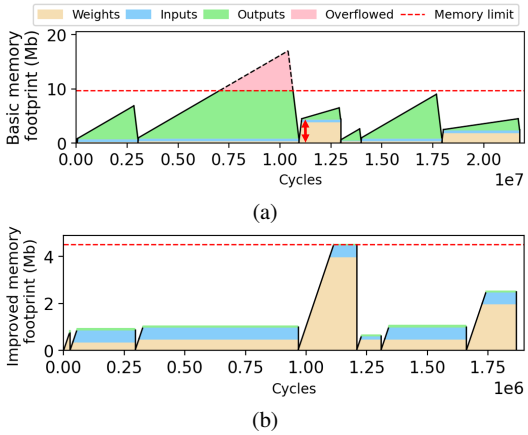


Fig. 10: On-chip memory footprint for one transformer block for: (a) the baseline architecture, and (b) the improved architecture.

transformer block as shown in Fig. 10(a). Note that overflow occurs during attention computation indicating that the memory management of the baseline architecture is not valid and needs to be improved.

Improved Architecture: To design an improved architecture, we employ IMCsim's run-time capability to determine what data (weights, inputs, and outputs) is immediately needed for MVM operations and which ones can be temporarily stored off-chip. The resulting reduction in memory requirements can be employed for additional IMC banks to enhance throughput.

First, we estimate the the maximum memory footprint for weights and inputs to be $4.48\,\text{Mb}$ (see red arrow at cycle no. 1.1×10^7 in Fig. 10(a)). Since each IMC bank generates a 128-bit (four 32-bit DPs) output, we get the smallest required memory footprint $M_{min} = 4.7 \times 10^6 + 128 \times N_{bank}$. Substituting M_{min} for M in (4) yields $N_{bank} \leq 67.2$. We choose $N_{bank} = 64$ to obtain $M_{min} = 4.49\,\text{Mb}$ which can be realized with 18 2048×128 SRAM banks.

The memory footprint (see Fig. 10(b)) of the improved architecture with 64 32×32 IMC banks and 18 2048×128 SRAM banks shows no memory overflow and a decrease in the overall latency by 90% due to the large number of IMC banks. Thus, this two-step analysis supported by IMCsim simulations is able to reveal a more effective architecture and memory management scheme.

Scheduling and Resource Allocation Analysis: IMCsim's cycle-level behavioral model enables detailed analysis of resource allocation and scheduling across time steps. The target is to identify and optimize underutilized resources. The resource allocation and scheduling of the improved architecture shown in Table V indicates that off-chip bus remains inactive after initial weights and inputs are loaded due to the single-port SRAM handling both off-chip and on-chip requests.

Table V also hints at the following optimizations to enhance utilization: 1) *memory optimization*: switching to dual-port SRAM can allow simultaneous off-chip reads and writes to reduce latency; partitioning SRAM into functional blocks (*e.g.*, input, weight, output SRAM) will help avoid SRAM bank conflicts; and 2) *computation optimization*: performing floating-point computations on-chip will reduce off-chip memory access.

Chip Implementation: We implemented the improved architecture in a $28\,\text{nm}$ process. We reused custom DIMC bank design from a recently completed chip design (to be reported in a separate publication). After verifying functionality in behavioral RTL with quantized software ground-truth, we synthesized the RTL using Genus, integrated the pre-designed DIMC bank layouts, and memory

TABLE V: Resource allocation and scheduling

Time Step	Off-chip Bus	SRAM	IMC Bank 0-63	ALU 0-63	Controller
1	Load	Write	Idle	Idle	Busy
2-24600	Load	Write	Idle	Idle	Idle
24601	Idle	Read	Write	Idle	Busy
24602-24632	Idle	Read	Write	Idle	Idle
24633	Idle	Read	Idle	Idle	Busy
24634-24645	Idle	Idle	MVM	Idle	Busy
24646	Idle	Write	Idle	Idle	Busy
24647	Idle	Read	Write	Idle	Busy
24648-24678	Idle	Read	Write	Idle	Idle
24679	Idle	Read	Idle	Idle	Busy
24680-24691	Idle	Idle	MVM	Idle	Busy
24692	Idle	Read	Idle	Idle	Busy
24693	Idle	Idle	Idle	Add	Busy
24694	Idle	Write	Idle	Idle	Busy
24695-24724	Idle	...	...	...	Busy
24725	Store	Read	Idle	Idle	Busy
24726-25044	Store	Read	Idle	Idle	Idle
25045-	...	...	...	...	...

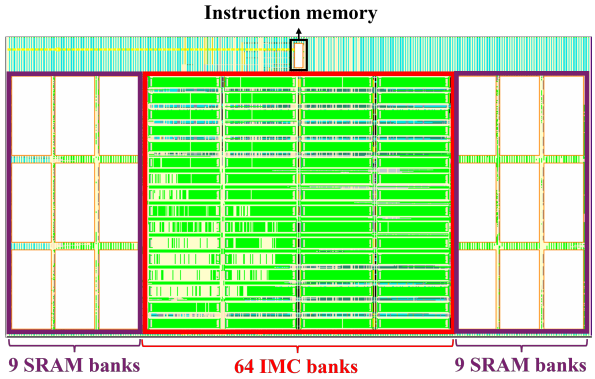


Fig. 11: A preliminary chip layout ($2.35\,\text{mm} \times 1.2\,\text{mm}$) in $28\,\text{nm}$ of an IMC processor for one transformer layer (pad ring omitted).

compiler-generated SRAM layouts. Next, we placed DIMC banks and SRAM banks (with a small instruction memory) and routed signals using Innovus, to generate a preliminary layout of the chip core implementing one transformer layer as shown in Fig. 11. This exercise demonstrates the use of IMCsim as a design tool.

V. Conclusion

This paper introduces IMCsim, a comprehensive, programmable, and extensible simulation framework and design tool for IMC systems. IMCsim is uniquely tailored to provide cycle-level simulation at the behavioral level, profiling run-time behavior, workload statistics, SNR, and other critical metrics necessary for designing IMC systems capable of executing large-scale deep learning workloads. To demonstrate its efficacy, we conducted detailed analyses and power, performance, area-aware design space evaluation which facilitate the IC design process. Our findings reveals new insights such as the impact of differing workload characteristics, the relation between quantization and utilization, the effect of off-chip access, and others. We demonstrated the capability and versatility of IMCsim, which positions it as a powerful tool for designing scalable IMC systems.

Acknowledgment

This work was supported by the Center for the Co-Design of Cognitive Systems (CoCoSys) funded by the Semiconductor Research Corporation (SRC) and the Defense Advanced Research Projects Agency (DARPA) under the JUMP 2.0 program.

REFERENCES

[1] T Andrulis, JS Emer, and V Sze. Cimloop: A flexible, accurate, and fast compute-in-memory modeling framework. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2024.

[2] Aayush Ankit et al. Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 715–731, 2019.

[3] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019.

[4] Rajeev Balasubramonian et al. Cacti 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Trans. Archit. Code Optim.*, 14(2), jun 2017.

[5] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, page 41, USA, 2005. USENIX Association.

[6] Hao Cai, Zhongjian Bian, Yaoru Hou, Yongliang Zhou, Jia-le Cui, Yanan Guo, Xiaoyun Tian, Bo Liu, Xin Si, Zhen Wang, Jun Yang, and Weiwei Shan. 33.4 a 28nm 2mb stt-mram computing-in-memory macro with a refined bit-cell and 22.4 - 41.5tops/w for ai inference. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 500–502, 2023.

[7] Yu-Der Chih, Po-Hao Lee, Hidehiro Fujiwara, Yi-Chun Shih, Chia-Fu Lee, Rawan Naous, Yu-Lin Chen, Chieh-Pu Lo, Cheng-Han Lu, Haruki Mori, Wei-Chang Zhao, Dar Sun, Mahmut E. Sinangil, Yen-Huei Chen, Tan-Li Chou, Kerem Akarvardar, Hung-Jen Liao, Yih Wang, Meng-Fan Chang, and Tsung-Yung Jonathan Chang. 16.4 an 89tops/w and 16.3tops/mm2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 252–254, 2021.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[9] facebookresearch. facebookresearch/dit: Inference llama 2 in one file of pure c.

[10] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

[11] An Guo, Xin Si, Xi Chen, Fangyuan Dong, Xingyu Pu, Dongqi Li, Yongliang Zhou, Lizheng Ren, Yeyang Xue, Xueshan Dong, Hui Gao, Yiran Zhang, Jingmin Zhang, Yuyao Kong, Tianzhu Xiong, Bo Wang, Hao Cai, Weiwei Shan, and Jun Yang. A 28nm 64-kb 31.6-tflops/w digital-domain floating-point-computing-unit and double-bit 6t-sram computing-in-memory macro for floating-point cnns. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 128–130, 2023.

[12] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[14] Shubham Jain et al. Rxnn: A framework for evaluating deep neural networks on resistive crossbars. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(2):326–338, 2021.

[15] Hongyang Jia et al. 15.1 a programmable neural-network inference accelerator based on scalable in-memory computing. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, volume 64, pages 236–238, 2021.

[16] Hongyang Jia, Hossein Valavi, Yinqi Tang, Jintao Zhang, and Naveen Verma. A programmable heterogeneous microprocessor based on bit-scalable in-memory computing. *IEEE Journal of Solid-State Circuits*, 55(9):2609–2621, 2020.

[17] Mingu Kang et al. An energy-efficient vlsi architecture for pattern recognition via deep embedding of computation in sram. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8326–8330. IEEE, 2014.

[18] Karpathy. karpathy/llama2.c: Inference llama 2 in one file of pure c.

[19] Anni Lu et al. Neurosim simulator for compute-in-memory hardware accelerator: Validation and benchmark. *Frontiers in Artificial Intelligence*, 4, 2021.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[21] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4195–4205, October 2023.

[22] Yubin Qin, Yang Wang, Dazheng Deng, Zhiren Zhao, Xiaolong Yang, Leibo Liu, Shaojun Wei, Yang Hu, and Shouyi Yin. Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[23] Satyabrata Sarangi and Bevan Baas. Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[24] Naresh R. Shanbhag and Saion K. Roy. Benchmarking in-memory computing architectures. *IEEE Open Journal of the Solid-State Circuits Society*, 2:288–300, 2022.

[25] Hugo Touvron et al. Llama: Open and efficient foundation language models, 2023.

[26] Fengbin Tu et al. A 28nm 15.59μj/token full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes. In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, volume 65, pages 466–468, 2022.

[27] Kodai Ueyoshi, Ioannis A. Papistas, Pouya Houshmand, Giuseppe M. Sarda, Vikram Jain, Man Shi, Qilin Zheng, Sebastian Giraldo, Peter Vrancx, Jonas Doevenspeck, Debjyoti Bhattacharjee, Stefan Cosemans, Arindam Mallik, Peter Debacker, Diederik Verkest, and Marian Verhelst. Diana: An end-to-end energy-efficient digital and analog hybrid neural network soc. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3, 2022.

[28] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017.

[29] Bo Wang, Chen Xue, Zhongyuan Feng, Zhaoyang Zhang, Han Liu, Lizheng Ren, Xiang Li, Anran Yin, Tianzhu Xiong, Yeyang Xue, Shengnan He, Yuyao Kong, Yongliang Zhou, An Guo, Xin Si, and Jun Yang. A 28nm horizontal-weight-shift and vertical-feature-shift-based separate-wl 6t-sram computation-in-memory unit-macro for edge depthwise neural-networks. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 134–136, 2023.

[30] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, apr 2009.

[31] Jinshan Yue, Chaojie He, Zi Wang, Zhaori Cong, Yifan He, Mufeng Zhou, Wenyu Sun, Xueqing Li, Chunmeng Dou, Feng Zhang, Huazhong Yang, Yongpan Liu, and Ming Liu. A 28nm 16.9-300tops/w computing-in-memory processor supporting floating-point nn inference/training with intensive-cim sparse-digital architecture. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 1–3, 2023.

[32] Qilin Zheng et al. Pimulator-nn: An event-driven, cross-level simulation framework for processing-in-memory-based neural network accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(12):5464–5475, 2022.

[33] Zhenhua Zhu, Hanbo Sun, Tongxin Xie, Yu Zhu, Guohao Dai, Lixue Xia, Dimin Niu, Xiaoming Chen, Xiaobo Sharon Hu, Yu Cao, Yuan Xie, Huazhong Yang, and Yu Wang. Mnsim 2.0: A behavior-level modeling tool for processing-in-memory architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(11):4112–4125, 2023.

Chameleon-SAT: An Adaptive Boolean Satisfiability Accelerator Using Mixed-Signal In-Memory Computing for Versatile SAT Problems

Iris Ying Chou, Hao Kong, Yi Huang, Jianfeng Zhu, Wenping Zhu, Shaojun Wei, Aoyang Zhang, and Leibo Liu
Tsinghua University, Beijing, China

***Abstract*—Boolean satisfiability (SAT), the first proven non-deterministic polynominal-complete problem, is crucial in data-intensive applications. Different applications have a wide spectrum of SAT problem sets (scale, complexity) and also various solution requirements (algorithm completeness, speed). Current SAT solvers are insufficient for providing ideal solutions under different scenarios. This work presents the Chameleon-SAT, the first ASIC-based SAT accelerator that can support local search, Davis-Putnam- Logemann-Lovel, Conflict-Driven Clause Learning algorithms, while leveraging the efficient mixed-signal in-memory computing architecture to achieve orders-of-magnitude improvements in speed compared to the prior SAT solvers. By judiciously selecting the reconfiguration mode, Chameleon-SAT is able to solve a wide range of the SAT problems to achieve small-scale, high-complexity cases ($\geq 90\times$ for 20 variables/ 86 clauses, satisfiable problems), medium-scale, structured cases ($\geq 19\times$ for 50 variables/ 215 clauses, unsatisfiable problems), and large-scale, high-complexity cases ($\geq 7\times$ for 100 variables/ 430 clauses, satisfiable problems).**

***Index Terms*—Boolean satisfiability, SAT solver, mixed-signal, processing-in-memory, accelerator, algorithm portfolio, adaptive accelerator.**

I. Introduction

Boolean satisfiability (SAT) is the problem of determining if an interpretation exists that satisfies a given Boolean formula. It plays a crucial role in computer science [1], [2], with far-reaching implications for various fields, such as logic circuit design [3], software verification, database management, artificial intelligence [4], bioinformatics [3], and cryptography [2]. The importance of this problem lies in the fact that it is directly related to the solvability and efficiency of solving complex problems in these fields [5].

There are two main types of algorithms for dealing with SAT problems [6]: incomplete algorithms, such as stochastic local search, and complete algorithms, such as Davis–Putnam–Logemann–Lovel (DPLL) [7], [8] and conflict-driven clause learning (CDCL) [9], [10]. Incomplete algorithms, such as stochastic algorithms [11], are known for their efficiency and flexibility. They are usually based on search and optimization strategies to quickly find possible solutions in a large solution space. Complete algorithms, such as DPLL and CDCL, focus more on the rigor and the precision of logical derivation. In the data–intensive applications, demand has surged for processing SAT problems, challenging algorithm speed and flexibility. The vast data volume, increased complexity, and diverse applications demand innovative computational methods and architectures.

Corresponding author: Aoyang Zhang (aoyang@tsinghua.edu.cn), and Leibo Liu (liulb@tsinghua.edu.cn).

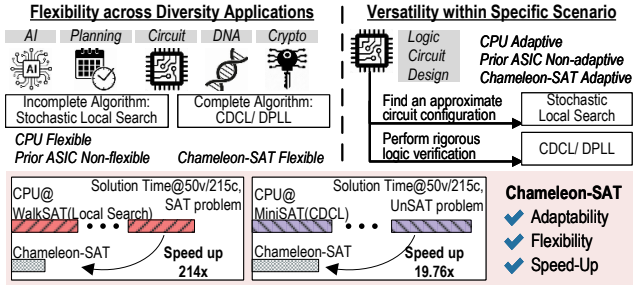
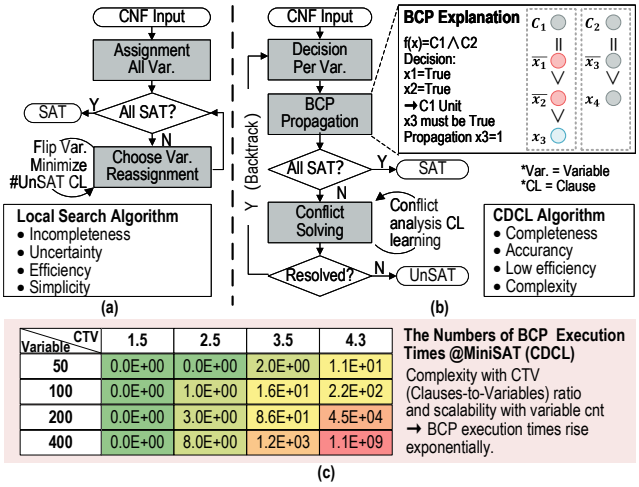


Fig. 1. Illustration of Chameleon-SAT solver's adaptability, flexibility, and speed across diverse applications, comparing CPU solution time and speedup for SAT/UnSAT problems using local search and CDCL methods.

First, SAT problems are widely applicable across diverse domains, and spanning these domains entails rapidly adjusting and optimizing algorithm execution to fit specific application scenarios and requirements. Supporting multiple algorithms and selecting the most appropriate ones for acceleration based on different contexts have become key to meeting this demand. Second, solving SAT problems requires different stochastic versus complete algorithms, even within specific identical scenarios. This is because different SAT algorithms exhibit very different solution times and hardware energy consumption. For example, in logic circuit design, local search–based algorithms can be employed to rapidly identify an approximate circuit configuration. To ensure that the circuit is correct and optimal, we then need to perform rigorous logic verification and optimization using deductive algorithms to ensure that the circuit meets all design specifications and performance requirements. However, current ASIC-based SAT accelerators support only one algorithm, making it hard to adapt to different requirements.

In response to these needs (as shown in Fig. 1), Chameleon-SAT is introduced. This mixed-signal in-memory computing accelerator not only supports externally designated algorithms but also features an adaptivity switch and internally selects the optimal algorithm for acceleration based on the characteristics and solution stages of the actual problem. The key contributions are summarized as follows.



Variable \ CTV	1.5	2.5	3.5	4.3
50	0.0E+00	0.0E+00	2.0E+00	1.1E+01
100	0.0E+00	1.0E+00	1.6E+01	2.2E+02
200	0.0E+00	3.0E+00	8.6E+01	4.5E+04
400	0.0E+00	8.0E+00	1.2E+03	1.1E+09

Fig. 2. (a) Incomplete algorithm (Local Search): Initializes, iteratively refines via heuristics, and verifies until solved or timed out. (b) Complete algorithm (CDCL): Initializes, constructs via decisions and BCP, backtracks, learns, and repeats until solved or proven unsolvable. (c) The numbers of BCP execution times across varying complexity and scalability.

- We designed Chameleon-SAT, the first SAT accelerator that supports both complete and incomplete SAT-solving algorithms (local search, CDCL, DPLL). By comparing previous ASICs that support only one algorithm (either complete or incomplete), Chameleon-SAT can adapt to various configurations to fit the requirements of different applications.
- We built an adaptive selection scheme that judiciously selects the optimal algorithm based on the defined quality function, thereby addressing the large variations (different CTV and variables) in a wide range of SAT problems.
- Chameleon-SAT utilizes a mixed-signal in-memory computing (IMC) architecture to enhance the level of speed in the hardware. Moreover, the data energy consumption is minimized thanks to the IMC architecture.
- Chameleon-SAT achieved small-scale, high-complexity cases ($\geq 90\times$ for 20 variables/ 86 clauses, satisfiable problems), medium-scale, structured cases ($\geq 19\times$ for 50 variables/ 215 clauses, unsatisfiable problems), and large-scale, high-complexity cases ($\geq 7\times$ for 100 variables/ 430 clauses, satisfiable problems).

II. Preliminary

A. Boolean SAT Problems

The SAT problem, focuses on determining whether a given Boolean expression is true. The problem can also be framed as a satisfiability challenge for propositional logic formulas and is commonly represented as a conjunctive normal form (CNF). In CNF, clauses are composed of ***OR***-connected literals, and the entire formula is connected by ***AND***. Literals can be positive or negative, representing the variable or its negation.

For example, consider the following CNF formula:

$$f = (x1 \vee \neg x2) \wedge (x2 \vee x3) \wedge (\neg x1 \vee \neg x3).$$

Here, we have three variables, $x1$, $x2$, and $x3$, and three clauses. Each clause is a disjunction (***OR***) of literals, and the entire formula is a conjunction (***AND***) of these clauses. The task is to find an assignment of 0 or 1 to each variable such that all the clauses evaluate to true simultaneously.

Depending on the number of literals in each clause, SAT problems are further categorized into k-SAT problems, such as 2-SAT or 3-SAT. In terms of computational complexity, SAT problems are proven to be nondeterministic polynominal complete, with 3-SAT serving as a typical example. Notably, in k-SAT problems, as the ratio of clauses to variables (CTV) changes, the problem's satisfiability undergoes a dramatic transition from being always satisfiable to being always unsatisfiable–a phenomenon known as the SAT phase transition. Specifically, in 3-SAT problems, a CTV of 4.3 is recognized as the critical threshold marking the transition from satisfiability to unsatisfiability, and problems at this threshold are defined as hard SAT problems.

B. SAT Algorithms

The algorithms for solving the Boolean satisfiability problem are classified into two main categories: complete search algorithms and incomplete stochastic local search algorithms.

Complete search algorithms, exemplified by DPLL and CDCL, constitute the classical approach. DPLL, known for its guarantee of completeness and correctness, offers a medium level of efficiency. It is particularly suited for small to medium-sized problems with moderate complexity, striking a balance between solution quality and processing speed. On the other hand, CDCL (illustrated in Fig. 2b), while incurring additional overhead due to clause learning, excels at solving large-scale, high-complexity problems with strict correctness requirements. Both DPLL and CDCL employ bifurcated branching search and integrate various optimization techniques, such as preprocessing, look-ahead, backtracking, nontemporal backjumping, and conflict-driven clause learning, to enhance search efficiency. Boolean constraint propagation (BCP), a core step in these algorithms, becomes exponentially more complex and scalable with increasing problem size (illustrated in Fig. 2c), thereby significantly impacting the solving process's efficiency.

In contrast, stochastic local search algorithms (shown in Fig. 2a), including GSAT, WalkSAT, and simulated annealing, focus on satisfying more clauses by continuously modifying assignments. Although they cannot prove unsatisfiability, they often swiftly find acceptable solutions. These algorithms are highly efficient and capable of rapidly processing large datasets, making them suitable for scenarios in which speed is crucial and a certain error rate is tolerable. Within this category, the efficiency of clause computation during each assignment modification is pivotal in reducing the overall solving time.

C. SAT Solvers Insufficiency

In the context of the significant data era, SAT solvers face unprecedented challenges. With the increasing size, complexity, and diversity of SAT problems, solvers need more vital

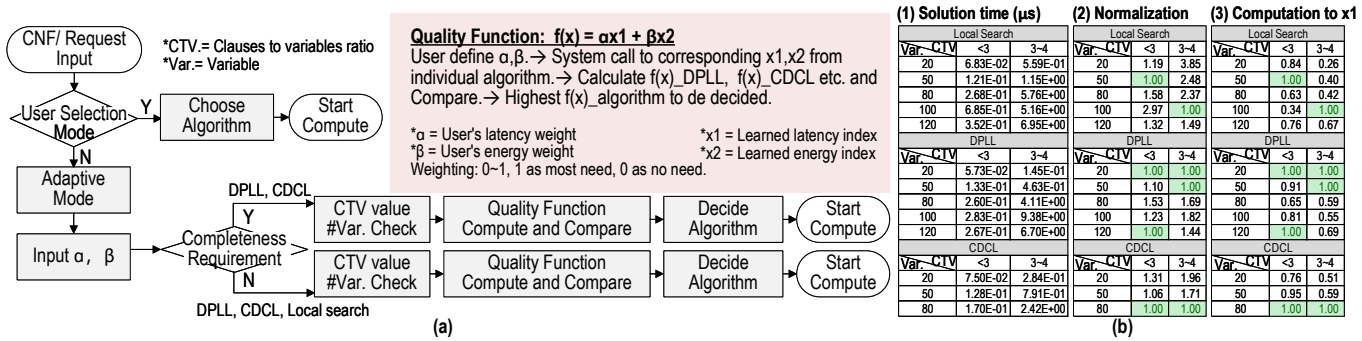


Fig. 3. Framework of Chameleon-SAT. (a) Flow of Chameleon-SAT operation. After inputting the SAT problem in CNF, the user can self-select an algorithm from a list to execution or adapt by specifying weights for solution time α, energy β, and completeness needs. Chameleon-SAT then chooses the optimal algorithm based on problem features (e.g., CTV and variable count) and a quality function score. (b) Depiction of the derivation of $x1$ in the quality function from test sets generated using conditional matrix analysis. This involves normalizing solution times and computing their reciprocals to obtain $x1$.

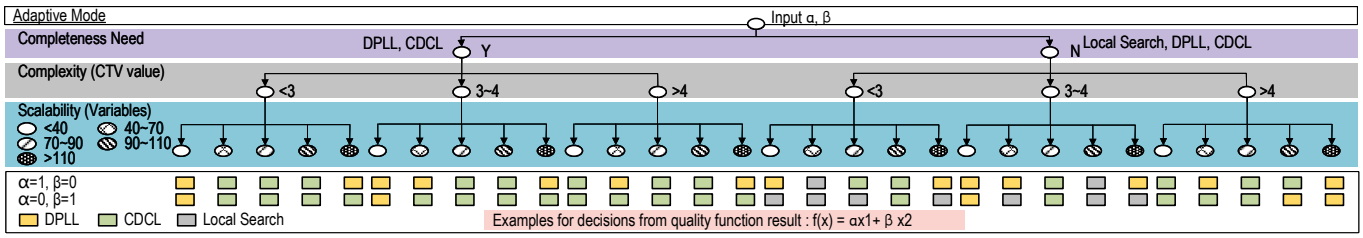


Fig. 4. Illustration of the branching process and examples of parameter settings in adaptive mode. After inputting the weights α (for solution time) and β (for energy), along with the completeness requirement, the system extracts CTV and the number of variables, categorizing the problems into 30 groups. Examples are provided for cases where $\alpha = 1$, $\beta = 0$, and $\alpha = 0$,$\beta = 1$, showcasing the most suitable algorithm with the highest quality function score for each group.

adaptability, more efficient algorithmic combining ability, and lower data processing overhead. However, current solvers still have significant deficiencies in these areas. To meet these needs, researchers have developed a variety of hardware [12] and software [13], [14] acceleration schemes. On the software side, solvers such as MiniSAT [15] are the mainstream in the industry, and Kissat [16] improves the solving capability by improving the data structure and algorithm optimization. ParaFROST [17] combines the CPU and GPU's computational power, significantly improving the performance of SAT solving through parallel processing. However, as the problem size increases, the data processing overhead becomes a pressing issue.

Hardware accelerators have emerged to overcome this limitation. Among them, FPGA-based solvers [18] have been implemented to also accelerate SAT-solving algorithms through customized hardware design. ASIC solutions, such as BRIM [19], Snap-SAT [20], and VIP-SAT [21], further improve the efficiency of solving problems by enhancing hardware functionality. Despite the performance improvement of these hardware accelerators, these architectures accelerate only one particular algorithm. In summary, recent SAT solvers have made some progress in hardware and software acceleration. However, they cannot support multiple SAT algorithms to meet the requirements of solvers in modern data-intensive applications.

III. Chameleon-SAT Framework

A. Overview and Operation Flow

Chameleon-SAT supports the user selection mode and the adaptive mode, as shown in Fig. 3a. First, the user must enter the CNF form of the SAT problem. Then, the system will ask the user if they want to use the self-selected algorithm mode.

(1) Self-selected algorithm mode: If the user selects yes, the system will provide a list of algorithms from which the specified algorithm can be selected. Once selected, the hardware will execute the appropriate algorithmic operations as instructed.

(2) Adaptive mode: Chameleon-SAT will enter the adaptive mode if the user chooses not to use the self-selected algorithm mode. In the adaptive mode, the user first needs to input the coefficient values of α and β, where α represents the weight of the solution time and α represents the weight of the energy. Immediately after that, the user needs to choose whether the solution's completeness is required. If completeness is required, Chameleon-SAT will choose the most suitable completeness algorithm (e.g., DPLL or CDCL). If a completeness requirement is unnecessary, the system will choose from algorithms including local search, DPLL, and CDCL. Next, the system extracts features of the problem, such as CTV and the number of variables. To determine the most suitable algorithm for a particular SAT problem, a quality function is defined. It considers both solution time and energy to decide the Chameleon-SAT's operation under different scenarios.

The quality function f(x) is the key to evaluating the merits of an algorithm:

$$f(x) = \alpha x1 + \beta x2.$$

Here, the α and β in the function are weighting coefficients given by the user, while $x1$ (solution time index) and $x2$ (energy index) are built from a library of already learned

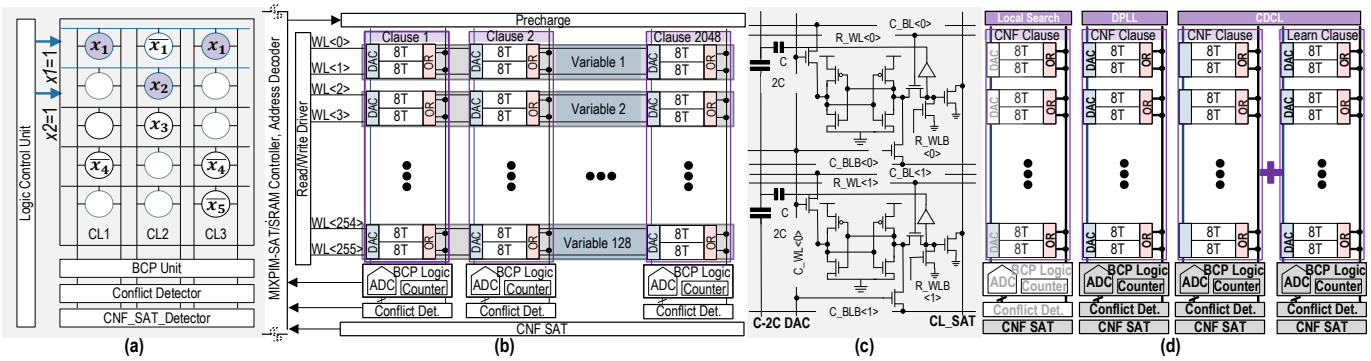


Fig. 5. (a) Chameleon-SAT architecture: Maps SAT to in-memory computing for parallel clause computation and BCP via WL activation. (b) SRAM array top-view: Columns = clauses, pairs = variables, literals stored, even/odd WLs assign 1/0. (c) SRAM cell pair: Represents variable, C-2C for DAC, CL-SAT for ***OR*** in clauses, illustrates DPLL/CDCL BCP. (d) PIM column usage in algorithms: Local search (***OR*** with CL-SAT), DPLL (BCP, conflict detection), CDCL (learning clauses for pruning).

data corresponding to the values of the variables for different algorithms at different scales (i.e., the number of variables) and at different complexities (e.g., CTV). The values of α and β are limited to a range between 0 and 1, and the sum must equal 1. When a weight is given to 0, it indicates that there is no demand for that index; when it is given to 1, it indicates that there is a strong demand for that index. For example, if α is given to 1 and β is given to 0, it means that the optimization goal is on solution time instead of energy, Chameleon-SAT will take only the solution time to be the criteria to determine the most suitable algorithm. By selecting different coefficients of α and β, the quality function can take into account both energy and the solution times of different SAT algorithms under the Chameleon-SAT solver. Moreover, even for the same SAT problem (same CTV and variables), the user may have different requirements of energy and latency. For example, on the server side, the solution time can be the priority to optimize, while on the terminal side, the energy consumption is more important. Depending on the user demand, Chameleon-SAT can determine the coefficient value to find the most suitable solution for a specific problem.

Fig. 3b shows how the $x1$ and $x2$ variables are obtained in the quality function. These variables are based on the results of a group of test sets, which are derived through conditional matrix analysis and the corresponding operations. To be specific, we first normalize the solution times of different algorithms under the same condition (same CTV and variables). After normalization, we calculate the reciprocal of each result to obtain the value of the $x1$ variable. For example, In Fig. 3b, when the variables are equal to 50 and the CTV is less than 3, local search, DPLL, and CDCL algorithms have solution times of 0.121 μs, 0.133 μs, and 0.128 μs, respectively. The normalization process takes the minimum solution time, 0.121 μs, to be the denominator. It normalizes the solution times to be 1, 1.1, and 1.06, respectively. The subsequent reciprocal process then converts the normalized number to between 0 and 1 for later quality function score calculation. Note that, the energy variable $x2$ uses the same methodology to calculate the corresponding value of each algorithm under the same condition. Finally, the system compares the quality function scores of the different algorithms, selects the algorithm with the highest score, and initiates the hardware to execute the corresponding algorithm for solving.

Fig. 4 details the exact process of parameter branching in adaptive mode and provides examples. In the adaptive mode, by inputting α and β weights and incorporating completeness requirements, Chameleon-SAT is able to determine the optimal solution under a given CTV as well as the number of variables. Currently, these variables are categorized into 30 different clusters. To illustrate the process more concretely, we provide two examples.

When α is equal to 1 and β is equal to 0, the solution time is mainly taken into account. This scenario is prepared for the server side, where the solution time can be the key parameter. Chameleon-SAT accelerator can determine the corresponding algorithms that need to be selected under different cases (different CTV and variables). Similarly, when α is equal to 0 and β is equal to 1, the energy consumption then becomes important. This scenario happens on the terminal side, where the energy consumption can be the important parameter that needs to be optimized. Within these different clusters, we look for the algorithms with the highest scores in the quality function that are considered the most suitable for their respective clusters.

B. Hardware Architecture

To accelerate SAT problem solving while maintaining low energy consumption in hardware design, we conducted a detailed analysis of the key steps and implemented targeted optimizations. Our focus was on two critical areas: the frequent clause matrix satisfiability operations following variable assignment flips in local search algorithms and the performance bottlenecks in the BCP steps of the CDCL and DPLL algorithms. We adopted IMC [22], [23], [24], [25], [26], [27] technology to address these core operations. IMC significantly reduces data handling overhead between the processor and memory by performing computations directly within the storage units. This leverages the parallel processing capabilities of the storage array, resulting in faster algorithm execution and lower energy consumption.

Fig. 5a illustrates how SAT problems are mapped and operated within the Chameleon-SAT IMC framework. By

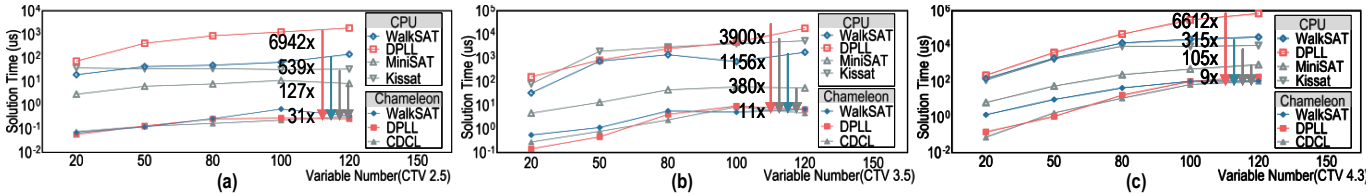


Fig. 6. Chameleon-SAT hardware accelerator outperforms CPU solvers in SAT problems.

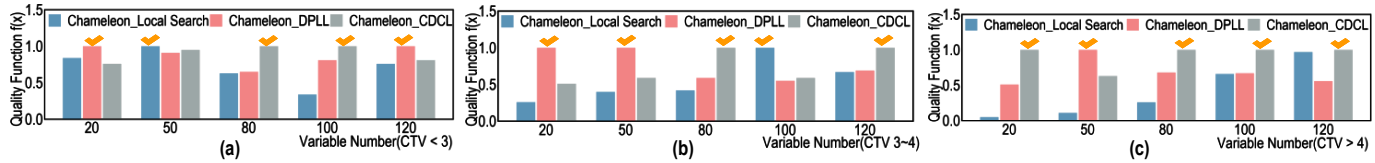


Fig. 7. Presentation of optimal SAT algorithms through quality function by scalability (variables) and complexity (CTV) with user assignment $\alpha = 1$, $\beta = 0$. Generally, DPLL suits small sizes (≤ 50 variables); for large (≥ 80 variables) and high complexity problems, CDCL and local search are best, with CDCL excelling in large-scale problems. CDCL for completeness; local search without.

activating the corresponding word lines (WLs) for variable assignments, we enable parallel computation of clauses in local search and CDCL/DPLL scenarios while supporting the BCP coding process. The top-view architecture (Fig. 5b) reveals the SRAM array matrix, in which each column represents a logical clause, and each pair corresponds to a logical variable. Literals (variables with signs) are stored within the clauses. WL activation is used to assign values to variables: even WLs represent variables assigned as 1, while odd WLs represent variables assigned as 0. Its mapping supports parallel internal clause computing.

Furthermore, we incorporated a mixed-signal design to tackle the complexity of variable identification coding in the BCP step. This design combines the precise control of digital circuits with the efficient processing power of analog circuits, thereby enhancing the variable identification coding process in BCP. Fig. 5c shows a schematic representation of an SRAM cell pair that serves as a variable with C-2C for digital-to-analog conversion (DAC) and CL-SAT for ***OR*** computation within the clause. Chameleon-SAT leverages the circuit design for the local search, DPLL, and CDCL algorithms to address computational bottlenecks, as illustrated in Fig. 5d.

In local search algorithm implementation, it several iterative flips are required to determine the satisfiability of CL-SAT. The proposed IMC architecture eliminates the read/write and movement of data between memory and the processor, enabling parallel operation of all clauses. Note that because there is no BCP operation in the local search algorithm, BCP logic and the conflict detection block are not activated.

In DPLL algorithm implementation, in addition to applying CL-SAT, the DPLL algorithm combines BCP logic and conflict detection functions for efficient propagation and conflict resolution. Thanks to the C-2C DAC, the BCP unit can operate in parallel by reading out the DAC output voltage, further enhancing the algorithm's performance.

Similar to DPLL, the CDCL algorithm further refines the solution process based on DPLL. It generates new learning clauses through learning and writes these clauses into SRAM to help constrain the search space in the next iteration to prune the solution space more efficiently.

IV. Evaluation

A. Experiments Setup

In the evaluation, Cadence Virtuoso was employed for SPICE simulations to assess the latency, power, and area of the in-memory computing unit. The digital components were designed in SystemVerilog and synthesized using Synopsys tools. All analog and digital designs and simulations were based on TSMC 28 nm technology. The in-memory computing unit was RTL abstracted for performance verification, ensuring accurate testing and reliability. We used Synopsys VCS and Verdi for simulation and debugging and benchmarked our results against CPU-based SAT solvers on an AMD Ryzen 9 9950X workstation.

B. Results and Analysis

1) Chameleon-SAT Performance Comparison with CPU Software: In a comparative study of solution times for SAT problems, Fig. 6 reveals the significant advantages of the Chameleon-SAT hardware accelerator over traditional CPU-based software solvers. The experimental results show that the hardware accelerator generally achieved 8.39–31.00-times speedup in solution times under various algorithmic operations. Especially in the CTV 2.5 clusters, when the problem size reached the maximum of 120 variables, the hardware accelerator achieved 31-fold acceleration, greatly improving the computational speed and efficiency. In addition, in the CTV 3.5 clusters, the hardware accelerator also performed well, achieving a 12-fold improvement in solution times, further proving its applicability in a wide range of application scenarios. In difficult SAT problem solving, such as CTV 4.3 hard SAT problems, the Chameleon-SAT accelerator was still able to maintain 8.39-times acceleration, demonstrating its stability and reliability in dealing with complex problems. In summary, the Chameleon-SAT hardware accelerator showed excellent performance compared with state-of-the-art SAT accelerators, providing acceleration for versatile SAT problems.

TABLE I
COMPARISON WITH THE STATE-OF-THE-ART SAT ACCELERATORS

	This Work Chameleon-SAT	**ASP DAC'21 HW-BCP [28]**	**Sci. Rep.'23 AIMS [29]**	**ISSCC'23 Snap-SAT [20]**	**ISSCC'24 VIP-SAT [21]**
Compute Model	ASIC Mixed-signal	CPU+ASIC Digital	ASIC Digital	ASIC Digital	ASIC Digital
Technology	28 nm CMOS	65 nm/7 nm CMOS	45 nm CMOS	65 nm CMOS	65 nm CMOS
In Memory Computation	Y	Y	Y	Y	Y
SRAM Access Rate	Low	Medium	Low	Low	Medium (Read Only)
Adaptability	Y	Y	N	N	N
Algorithm	Complete& Incomplete DPLL/ CDCL/ Local Search	Complete MiniSAT2	Incomplete Ising Machine	Incomplete WalkSAT	Incomplete Stochastic
Test Case	Hard 3-SAT (CTV = 4.30)	N/A N/A	Hard 3-SAT (CTV = 4.30)	Hard 3-SAT (CTV = 4.30)	Hard 3-SAT (CTV = 4.55)
Max. Number of Var	128	N/A	500	128	50
Max. Number of Cla	2048	N/A	1917	1024	218
Solution Time	1.1 μs (50-var/ 215-cla)	N/A N/A	1.329 μs (50-var/ 218-cla)	713 μs (60-var/ 258-cla)	18.7 μs (50-var/ 218-cla)
Energy	2.849 nJ[a] **(50-var/ 215-cla)**	N/A N/A	N/A N/A	1098 nJ (60-var/ 258-cla)	20.8 nJ (50-var/ 218-cla)
Area	18 mm^2	4 cm^2/ 3.3 cm^2	169 mm^2	0.93 mm^2	1.116 mm^2

[a]Energy measured for PIM (Processing-in-Memory) array only.

2) Chameleon-SAT Performance Comparison with ASIC Hardware: A comparative analysis of Chameleon-SAT and the emerging research on SAT [20], [21], [28], [29] from recent years, as shown in TABLE I. Our SAT hardware accelerator demonstrated performance enhancements on several key performance metrics (solution time and energy consumption) and was the only accelerator capable of supporting all three algorithms simultaneously. A detailed comparison is provided below.

Technology node: We used 28 nm CMOS technology, offering inherent advantages in power and performance compared to 45/ 65 nm counterparts. In-memory computation: Chameleon-SAT supported in-memory computation, a key technology for improving computational efficiency, while other devices did not. Algorithm support: Our hardware accelerator was the only device that supported DPLL, CDCL, and Ising Machine algorithms, demonstrating the design's strength in algorithmic versatility and flexibility. Test case: Focusing on high-CTV 3-SAT problems validates its ability to resolve complex logic constraints. Scalability: Chameleon-SAT can handle up to 500 variables and 2048 clauses, the highest among the compared devices, thereby demonstrating its ability to tackle large-scale problems. Solution time: Our appliance solved a problem with 50 variables and 218 clauses in only 9.91 μs, the fastest among the compared devices. Area: An area of 18 mm² reflects the trade-off for supporting advanced functionalities at scale. In summary, our SAT hardware accelerator demonstrated excellent performance in terms of algorithm support, efficiency, energy efficiency, and scalability.

3) Chameleon-SAT Quality Function for Adaptive Mode: To build up the adaptive mode, we conducted various test cases to build up the quality function needs. The results illustrated and presented in Fig. 7 for our quality function with $\alpha = 1$ and $\beta = 0$, show the optimal algorithm choices for different problem sizes and complexities. The check marks indicate the highest-scoring, most-suitable algorithms for specific conditions of size and complexity.

Generally, DPLL emerged as a preferred choice for solving time for smaller problem sizes, such as 50 variables, even with low to medium complexity. However, as the problem size increased beyond 80 variables, local search and CDCL became more appropriate algorithms, particularly under high-complexity conditions. Notably, CDCL excelled and scored significantly higher, demonstrating its suitability for solving large-scale SAT problems with high complexity.

Furthermore, when completeness is required, CDCL can meet this demand while still performing well. Conversely, when completeness is not a requirement, local search becomes a viable option.

V. CONCLUSION

This work presents Chameleon-SAT, a mixed-signal in-memory computing accelerator that enhances energy consumption and solution time performance in Boolean satisfiability solving, offering an adaptive solution to solving SAT problems.

It is the first accelerator to support both complete and incomplete algorithms, efficiently handling a wide range of SAT problem scales and complexities. This results in substantial speed improvements over traditional and emerging solvers, making it a promising solution for diverse data-intensive applications. It achieves small-scale, high-complexity cases ($\geq 90\times$ for 20 variables/ 86 clauses, satisfiable problems), medium-scale, structured cases ($\geq 19\times$ for 50 variables/ 215 clauses, unsatisfiable problems) and large-scale, high-complexity cases ($\geq 7\times$ for 100 variables/ 430 clauses, satisfiable problems).

ACKNOWLEDGMENT

This work is supported in part by the National Key R&D Program of China (Grant No. 2024YFB3108102), and in part by the National Natural Science Foundation of China (Grant No. U22B2024).

REFERENCES

[1] S. Cook, "The complexity of theorem proving procedures," in Proc. 3rd Annu. ACM Symp. Theory Comput., 1971, pp. 151–158.

[2] J. Gu, "Global optimization for satisfiability (SAT) problem," IEEE Trans. Knowl. Data Eng., vol. 6, no. 3, pp. 361–381, 1994.

[3] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD), 2017.

[4] M. S. Chowdhury, J. You, C. Chen, and D. Johnson, "Guiding CDCL SAT search via random exploration amid conflict depression," in Proc. AAAI Conf. Artif. Intell., vol. 34, no. 2, 2020, pp. 1428-1435.

[5] S. Yu, M. Liu, W. Dou, X. Liu, and S. Zhou, "Networking for big data: A survey," IEEE Commun. Surv. Tutor., vol. 19, no. 1, pp. 531–549, 2016.

[6] H. Kautz, A. Sabharwal, and B. Selman, "Incomplete algorithms," in Handbook of Satisfiability. Amsterdam, Netherlands: IOS Press, 2009, pp. 185–203.

[7] M. Davis and H. Putnam, "A computing procedure for quantification theory," J. ACM, vol. 7, no. 3, pp. 201–215, Jul. 1960. [Online]. Available: https://doi.org/10.1145/321033.321034

[8] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Abstract DPLL and abstract DPLL modulo theories," in Proc. Int. Conf. Logic Program. Artif. Intell. Reason. (LPAR), 2004, pp. 36–50.

[9] M. Luo, C.-M. Li, F. Xiao, F. Manya, and Z. Lu, "An effective learnt clause minimization approach for CDCL SAT solvers," in Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI), 2017.

[10] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," in Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD), 2001, pp. 279–285.

[11] B. Selman, H. A. Kautz, and B. Cohen, "Local search strategies for satisfiability testing," in Cliques, Coloring, Satisfiability, vol. 26, DIMACS, 1993, pp. 521–532.

[12] D. Kim, N. M. Rahman, and S. Mukhopadhyay, "Presto: A processing-in-memory-based k-SAT solver using recurrent stochastic neural network with unsupervised learning," IEEE J. Solid-State Circuits, vol. 59, no. 7, pp. 2310–2320, 2024.

[13] G. Audemard and L. Simon, "On the glucose SAT solver," Int. J. Artif. Intell. Tools, vol. 27, no. 1, pp. 1–25, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49573977

[14] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in Proc. 38th Des. Autom. Conf., 2001, pp. 530–535.

[15] N. Een and N. Sorensson, "An extensible SAT-solver," in Theory Appl. Satisfiability Test., E. Giunchiglia and A. Tacchella, Eds. Berlin, Germany: Springer, 2004, pp. 502–518.

[16] M. Fleury and A. Biere, "Mining definitions in Kissat with Kittens," Form. Methods Syst. Des., vol. 60, no. 1, pp. 381–404, 2022. [Online]. Available: https://doi.org/10.1007/s10703-023-00421-2

[17] M. Osama, A. Wijs, and A. Biere, "Certified SAT solving with GPU accelerated inprocessing," Form. Methods Syst. Des., vol. 62, no. 1, pp. 79–118, 2024. [Online]. Available: https://doi.org/10.1007/s10703-023-00432-z

[18] G.-J. Nam, K. A. Sakallah, and R. A. Rutenbar, "Satisfiability-based layout revisited: Detailed routing of complex FPGAs via search-based Boolean SAT," in Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA), Monterey, CA, USA, 1999.

[19] R. Afoakwa, Y. Zhang, U. K. R. Vengalam, Z. Ignjatovic, and M. Huang, "BRIM: Bistable resistively-coupled Ising machine," in Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA), 2021, pp. 749–760.

[20] S. Xie et al., "SNAP-SAT: A one-shot energy-performance-aware all-digital compute-in-memory solver for large-scale hard Boolean satisfiability problems," in Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC), Feb. 2023, pp. 420–422.

[21] C. Shim, J. Bae and B. Kim, "30.3 VIP-Sat: A Boolean Satisfiability Solver Featuring 5×12 Variable In-Memory Processing Elements with 98% Solvability for 50-Variables 218-Clauses 3-SAT Problems," 2024 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2024, pp. 486-488, doi: 10.1109/ISSCC49657.2024.10454397.

[22] D. Patterson et al., "A case for intelligent RAM," IEEE Micro, vol. 17, no. 2, pp. 34–44, 1997.

[23] M. Oskin, F. Chong, and T. Sherwood, "Active pages: A computation model for intelligent memory," in Proc. 25th Annu. Int. Symp. Comput. Archit., 1998, pp. 192–203.

[24] J. Draper et al., "The architecture of the DIVA processing-in-memory chip," in Proc. 16th Int. Conf. Supercomput. (ICS), 2002.

[25] D. Elliott, M. Stumm, W. Snelgrove, C. Cojocaru, and R. McKenzie, "Computational RAM: Implementing processors in memory," IEEE Des. Test Comput., vol. 16, no. 1, pp. 32–41, 1999.

[26] S. Yin et al., "Monolithically integrated RRAM- and CMOS-based in-memory computing optimizations for efficient deep learning," IEEE Micro, vol. 39, no. 6, pp. 54–63, 2019.

[27] H. Wang et al., "A 32.2 TOPS/W SRAM compute-in-memory macro employing a linear 8-bit C-2C ladder for charge domain computation in 22nm for edge inference," in Proc. IEEE Symp. VLSI Technol. Circuits, 2022, pp. 36–37.

[28] S. Park, J.-W. Nam, and S. K. Gupta, "HW-BCP: A custom hardware accelerator for SAT suitable for single chip implementation for large benchmarks," in Proc. Asia South Pac. Des. Autom. Conf. (ASP-DAC), 2021, pp. 29–34.

[29] A. Sharma, M. Burns, A. Hahn, and M. Huang, "Augmenting an electronic Ising machine to effectively solve Boolean satisfiability," Sci. Rep., vol. 13, no. 1, p. 22858, 2023.

BBAL: A Bidirectional Block Floating Point-Based Quantisation Accelerator for Large Language Models

Xiaomeng Han[1*], Yuan Cheng[2,3*†], Jing Wang[1], Junyang Lu[1], Hui Wang[1], X.x. Zhang[4], Ning Xu[1], Dawei Yang[2✉], Zhe Jiang[1✉]
[1]National Center of Technology Innovation for EDA, School of Integrated Circuits, Southeast University. [2]Houmo AI.
[3]Nanjing University. [4]Jilin Normal University.
mingzhihan7@gmail.com, yuancheng@smail.nju.edu.cn, nick585108@163.com, 2972462961@qq.com,
whmio0115@seu.edu.cn, 3255594256@qq.com, xning@seu.edu.cn, dawei.yang@houmo.ai, zhejiang.uk@gmail.com

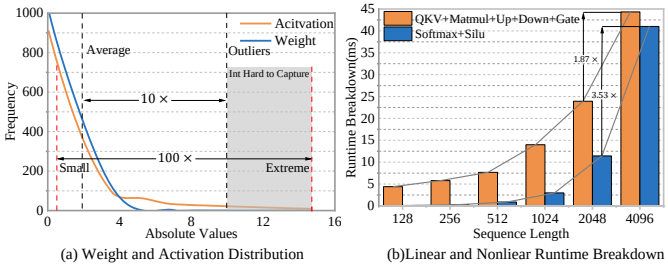
***Abstract*—Large language models (LLMs), with their billions of parameters, pose substantial challenges for deployment on edge devices, straining both memory capacity and computational resources. Block Floating Point (BFP) quantisation reduces memory and computational overhead by converting high-overhead floating point operations into low-bit fixed point operations. However, BFP requires aligning all data to the maximum exponent, which causes loss of small and moderate values, resulting in quantisation error and degradation in the accuracy of LLMs. To address this issue, we propose a Bidirectional Block Floating Point (BBFP) data format, which reduces the probability of selecting the maximum as shared exponent, thereby reducing quantisation error. By utilizing the features in BBFP, we present a full-stack Bidirectional Block Floating Point-Based Quantisation Accelerator for LLMs (BBAL), primarily comprising a processing element array based on BBFP, paired with proposed cost-effective nonlinear computation unit. Experimental results show BBAL achieves a 22% improvement in accuracy compared to an outlier-aware accelerator at similar efficiency, and a 40% efficiency improvement over a BFP-based accelerator at similar accuracy.**

I. Introduction

LLMs [1] have achieved remarkable success, including text generation [2], text understanding [3], and language translation [4]. However, the substantial size and high computational cost of these models constrain their deployment on edge devices. For instance, deploying a Llama-70B [5] model necessitates the use of at least two 80GB A100 GPUs. With the continuous growth in the size of models, deploying them on edge devices imposes higher overhead. Thus, reducing computational cost and storage demands has emerged as a critical challenge that needs to be addressed [6], [7].

Various techniques have been investigated [8] to facilitate the efficient deployment of LLMs. Among them, quantisation [9]–[11] is one of the most effective methods for reducing inference cost. Especially, INT quantisation has been proven to significantly reduce storage requirements and computational cost when compared to the floating point. Yet, due to the limitations imposed by the representational range of INT and the presence of outliers in LLMs (see Fig. 1 (a)), INT quantisation usually faces significant accuracy degradation. For instance, INT8 [12] quantisation increases perplexity (PPL) over 20%.

To address this issue, researchers have proposed several data formats with a wider dynamic range to capture the outliers; examples include BF16 [13], FP8 [14], etc. Among these, Block Floating Point (BFP) stands out as a promising quantisation method [15]–[18], offering a favourable trade-off between performance and hardware overhead. By compelling a set of fixed point numbers to share a common exponent, BFP effectively converts floating point operations into fixed point computations. Current research on BFP primarily focuses on accelerating computations in linear layers [19]–[21], and the nonlinear layers are entirely ignored. However, Transformer-based models [22], contain numerous nonlinear operations, including Softmax and SILU, which often involve transcendental function



(a) Weight and Activation Distribution (b)Linear and Nonliear Runtime Breakdown

Fig. 1: (a) Distribution of activation and weight values in OPT-6.7B. (b) Linear and nonlinear runtime in the decoder stage of Llama-7B.

computations and exhibit lower robustness, typically necessitating a broader representation range, high-precision, and higher-overhead floating point computations. As revealed in Fig. 1 (b), the computation time taken by nonlinear operators increases with the growth of input sequence length, gradually becoming a performance bottleneck [23]. Since BFP offers a similar representation range to floating point, with greater computational efficiency, it holds the potential for simplifying operations in nonlinear layers. However, BFP's strategy of aligning all data to the maximum exponent introduces moderate quantisation error. This limits its ability to achieve lower-bit BFP quantisation in linear layers and hinders its application in nonlinear computations. For instance, directly applying BFP4 [15] quantisation to the linear layers of the OPT [24] model can result in over a 40% increase in PPL. Similarly, as shown in Table IV, using BFP for computations in nonlinear layers can lead to a $5\times$ increase in PPL.

Contributions: To reduce the quantisation error caused by BFP's alignment strategy and explore its potential applications in nonlinear layers, we propose a block-based data format, Bidirectional Block Floating Point (BBFP). By employing a 1-bit flag to distinguish between high and low mantissas, not all data needs to be aligned to the maximum exponents, reducing the probability of selecting a larger exponent as the shared exponent. This enables BBFP to capture outliers while minimizing quantisation error for moderate and small values. Finally, based on BBFP, we propose BBAL, primarily comprising an optimized PE array based on BBFP, paired with our proposed cost-effective nonlinear computation unit. Experimental results show that BBAL achieves a 22% improvement in accuracy compared to an outlier-aware accelerator at similar efficiency and a 40% efficiency improvement over a vanilla BFP-based accelerator at similar accuracy. The main contributions presented are as follows:

- **Data Format:** We propose BBFP to reduce BFP quantisation error. We also explore the impact of different shared exponent selections and overlap bit configurations on the quantisation error of BBFP. Finally, based on the characteristics of BBFP, we design an efficient basic computation unit.
- **Nonlinear Unit:** Leveraging the low quantisation error and

✉Corresponding authors.
*Equal contribution.
†This work was conducted during his internship at Houmo AI.

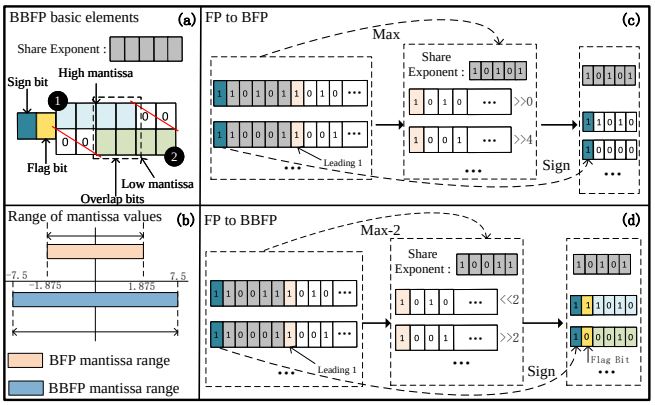


Fig. 2: (a) The basic components of BBFP(4,2); (b) comparing the representational range of mantissas between BBFP and BFP; (c) the FP to BFP process; (d) the FP to BBFP process.

fixed point computation characteristics of BBFP, we propose an efficient nonlinear computation unit based on BBFP. Additionally, benefiting from the shared exponent feature of BBFP, we introduce an exponent-based segmented lookup table method, which improves compatibility and reduces consumption.

- **LLMs Accelerator:** Finally, we present BBAL, primarily comprising an optimized PE array based on BBFP for efficient linear computation, paired with our proposed cost-effective nonlinear computation unit to reduce resource consumption and latency.

II. Background

A. Quantisation

Currently, quantisation data formats focus on lower-bit formats, such as floating point FP4 [25], and fixed point like INT4 [26], both of which provide high memory efficiency. However, due to the lower arithmetic density of floating point, researchers prefer using fixed point formats for quantisation. Methods like SmoothQuant [27] and GPTQ [28] have effectively utilized fixed point for quantisation. Nonetheless, the inherent limitations of integer representations and the sensitivity to outliers present significant challenges in achieving high performance with lower-bit fixed point quantisation.

Quantisation methods can generally be categorized into two approaches: Post-Training Quantisation (PTQ) [29] and Quantisation-Aware Training (QAT) [30]. While QAT can maintain better accuracy at the same data width, the substantial model size of LLMs results in significant training cost, making PTQ a concise and effective quantisation method. In this work, we adopt the PTQ and propose BBFP that allows weight-activation quantisation without calibration.

B. Block Floating Point

In the IEEE-754 standard, each single-precision floating point number is composed of three parts: a 1-bit sign s, an 8-bit exponent e, and a 23-bit mantissa m. The actual value represented by the floating point number can be summarized as: $v = (-1)^s \times 1.m \times 2^{e-e_{bias}}$. Thus, for a vector of elements, the floating point representation is:

$$[(-1)^{s_0}2^{e_0}m_0, (-1)^{s_1}2^{e_1}m_1, ..., (-1)^{s_{n-1}}2^{e_{n-1}}m_{n-1}] \quad (1)$$

An format for optimizing performance and improving memory density is Block Floating Point (BFP). As shown in Fig. 2 (c), BFP within a block share a max exponent by shifting the mantissa, and can be expressed as follows:

$$2^{e_m}[(-1)^{s_0}m'_0, (-1)^{s_1}m'_1, ..., (-1)^{s_{N-1}}m'_{N-1}] \quad (2)$$

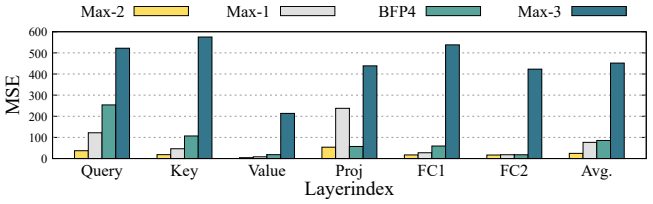


Fig. 3: Comparison of the impact of different selection of shared exponent of activation quantisation error with BBFP (4,2).

Where 2^{e_m} is the maximum exponent within a block of data, and m'_i is the shifted mantissa. Thus, the dot product of two vectors in BFP format can be expressed as follows:

$$2^{e_{m_1}+e_{m_2}} \sum_{i=0}^{N-1} ((-1)^{s_{1,i} \oplus s_{2,i}} m_{1,i} \cdot m_{2,i}) \quad (3)$$

where e_{m_1} and e_{m_2} are the shared exponent of two BFP vectors, $\oplus$ is an XOR operation.

By transforming the complex floating point dot product operation into a fixed point equivalent, BFP significantly enhances computational efficiency. However, this improvement comes at the cost of reduced precision for small and moderate values, leading to a modest reduction in accuracy when quantizing LLMs.

III. The Proposed Data Format

A. BBFP Data Format

BBFP Definition: To reduce precision loss in small and moderate values during BFP computations, we propose a data format named BBFP. Fig. 2 (a) illustrates the proposed BBFP data format, which consists of a sign bit, a flag bit, e bits for the shared exponent, m bits for the mantissa, and o bits for the overlap. Throughout this paper, we denote different configurations of BBFP as BBFP(m,o) and different mantissa bit-width of BFP as BFPm, where $m, o \in \mathbb{N}$. In all configurations, the shared exponent bit-width is fixed at 5 bits.

In BBFP, the 1-bit flag serves to indicate whether the mantissa should be left- or right-shifted during alignment, while the o bits of overlap are used to reduce error introduced by truncation when shifting left. The FP16 with an 11-bit mantissa and implicit leading one to BBFP(4,2) conversion can be summarized as follows:

$$x_{\text{BBFP(4,2)}} = \begin{cases} Clip(x << n)_{13,10}\,, \text{Flag} = 1 \\ Clip(x >> n)_{11,8}\ \ , \text{Flag} = 0 \end{cases} \quad (4)$$

where $Clip(\cdot)_{13,10}$ and $Clip(\cdot)_{11,8}$ represent truncating the original mantissa from bit 13 to bit 10 and from bit 11 to bit 8. n represents shift count and Flag represents a 1-bit flag in BBFP. Due to the presence of the 2-bit overlap, truncation does not begin from the 12th bit to distinguish between left- (high) and right-shifted (low) mantissas, but rather from the 10th bit. Regardless of whether the mantissa is shifted left or right, it is truncated to 4 bits and stored in memory. For example, when converting FP16 to BBFP without overlap bits, the most significant bit of the original mantissa is stored in the high-mantissa group, while the remaining bits are truncated and discarded. However, with the addition of two overlap bits, truncation starts from the 10th bit of the original mantissa, preserving 3 bits of information and thereby reducing quantisation error.

Fig. 2 (d) shows the process of converting FP to BBFP in detail, which is similar to FP to BFP shown in Fig. 2 (c). First, a shared exponent is determined, using the $Max - 2$ exponent as an example. Next, the origin exponent is compared with the shared exponent and the origin mantissa is shifted. If the original floating point exponent is greater than the shared exponent, the flag is set to 1, and the mantissa

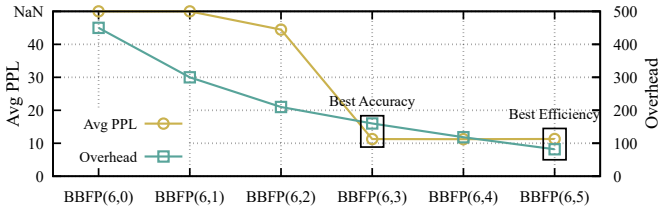


Fig. 4: The PPL and hardware overhead for BBFP with a width of 6 under varying overlap bit-widths.

is left-shifted. When the floating point exponent is less than or equal to the shared exponent, the flag is set to 0, and the mantissa is either right-shifted or remains unchanged. Finally, the shifted mantissa is truncated to four bits. Thus, the value represented by BBFP can be summarized by the following equation:

$$2^{e_s}[(-1)^{s_0}m'_0 \times f, (-1)^{s_1}m'_1 \times f, ..., (-1)^{s_{N-1}}m'_{N-1} \times f] \quad (5)$$

$$f = \begin{cases} 1, & \text{Flag} = 0 \\ 2^{m-o}, & \text{Flag} = 1 \end{cases} \quad (6)$$

Where e_s represents the shared exponent, m'_i represents the shifted mantissa, Flag indicates the flag bit, m represents the bit width of the mantissa, and o represents the bit width of the overlap. From the above equation, it is evident that, given the same mantissa width, BBFP provides enhanced representational capability for the mantissa compared to BFP, as illustrated in Fig. 2 (b).

BBFP Dot Product: The dot product of two vectors in BBFP format can be expressed as follows:

$$2^{e_{s_1}+e_{s_2}} \sum_{i=0}^{N-1} ((-1)^{s_{1,i} \oplus s_{2,i}} m_{1,i} f_{1,i} \cdot m_{2,i} f_{2,i}) \quad (7)$$

the above expression demonstrates that BBFP retains the characteristic of BFP that converts floating-point operations into fixed-point operations. By incorporating multiplexer and shifting modules, the mantissa representational range increases by 4×.

B. Quantisation Error Analysis

To compare the quantisation error with BFP, we analyse the sources of error in block floating point quantisation. For a block floating point using the round to nearest scheme, its quantisation error is zero-mean, and its variance σ^2 can be described as follows [31]:

$$\sigma^2 = \frac{2^{-2L_m}}{12} \sum_{i=1}^{N_\gamma} p_{\gamma_i} 2^{2\gamma_i} \quad (8)$$

Where L_m denotes the length of the mantissa, and p_{γ_i} represents the probabilities mass function of the block exponent. $N_\gamma = 2^{L_E}$ is the number of available block exponent levels. When L_m are the same for both BFP and BBFP, the only factor influencing the quantisation error is p_{γ_i}. Compared to the operation of aligning to the maximum exponent in BFP, BBFP allows for alignment to the non-maximum exponent, decreasing the quantisation error variance.

C. Selection of Shared Exponent

As described in Eq. (4), converting FP16 to BBFP involves shifting and truncating the mantissa. Intuitively, higher bits of data have more significance, so protecting these higher bits during the shift and truncation process can reduce quantisation error. However, as shown in Eq. (8), reducing the probability of selecting the maximal shared exponent can minimize quantization error caused by right shift and truncation. Therefore, based on the two factors mentioned above, the selection of shared exponent we propose is as follows:

$$E_{\text{shared}} = Max(E) - (m - o) \quad (9)$$

where Max denotes taking the maximum value, m represents the width of the mantissa, o and represents the width of the overlap bits. Fig. 3 compares the quantisation error of BBFP with a 4-bit mantissa and different shared exponent and BFP4. The $max - 3$ alignment strategy, which is defined as $max-(m-o)-1$, results in significant error due to the left shift of the most significant bit, moving it out of the truncation range. By contrast, the $max - 1$ alignment strategy, defined as $max - (m - o) + 1$, is more likely to select larger values as the shared exponent compared to the $max - (m - o)$ alignment strategy, leading to more error.

D. Selection of Overlap Bit Width

When the shared exponent selection strategy is set, increasing the width of overlap bit can reduce error from truncation due to left shift. However, according to Eq. (9) a wider overlap increases the probability of encountering the maximal shared exponent, which leads to greater loss for moderate and small values. Moreover, since various LLMs exhibit distinct data distribution and sensitivities to numerical error, and different overlap widths result in different hardware cost, we propose Algo. 1 to determine the overlap widths.

By adjusting the overhead weight in Algo. 1, we can balance the priorities of accuracy and hardware cost. Fig. 4 shows the process of optimizing the overlap width for BBFP with a width of 6.

IV. Data Format-Driven Hardware Design

A. Linear Computing Unit

The Multiply and Accumulate (MAC) module is a fundamental computational unit in LLMs, primarily involving inter-block multiplication and partial-sum addition. Given that the proposed data format exhibits structured bit-level sparsity during computation, we optimize the MAC module to improve computational density.

Intra-Block Multiplication: As shown in Fig. 5 (a), the inter-block matrix multiplication in BBFP can be divided into shared exponent addition and block mantissa multiplication. The mantissa multiplication can be expressed as follows:

$$m_0 = \begin{cases} m_1 \times m_2 & , \text{Flag}_1 \& \text{Flag}_2 = 0 \\ m_1 \times m_2 << 2 & , \text{Flag}_1 \oplus \text{Flag}_2 = 1 \\ m_1 \times m_2 << 4 & , \text{Flag}_1 \& \text{Flag}_2 = 1 \end{cases} \quad (10)$$

Where F_1,F_2 represent the two Flag bits. In the case of BBFP(4,2), the multiplication of the two 4-bit mantissas can be performed using

Algorithm 1 Selection of overlap bit width

Input: LLMs $Model$, Weight of overhead w, Width of Mantissa m.
Output: Optimal Overlap Bits o.

Function `Select_Best_Width`($Model$, w, m)
for $i = 0$ to $m - 1$ **do**
    $PPL[i]$ =Calculate_PPL($model$,BBFP(m,i))
    $Overhead[i]$ =Calculate_Overhead(BBFP(m,i))
end for
for $j = 0$ to $m - 1$ **do** ▷ #Max Norm and Cal score
    $PPL[j] = PPL[j]/Max(PPL)$
    $Overhead[j] = Overhead[j]/Max(Overhead)$
    $score[j] = w \times Overhead[j] + (1 - w) \times PPL[j]$
end for
$o = Min(score).index$
return o
End Function

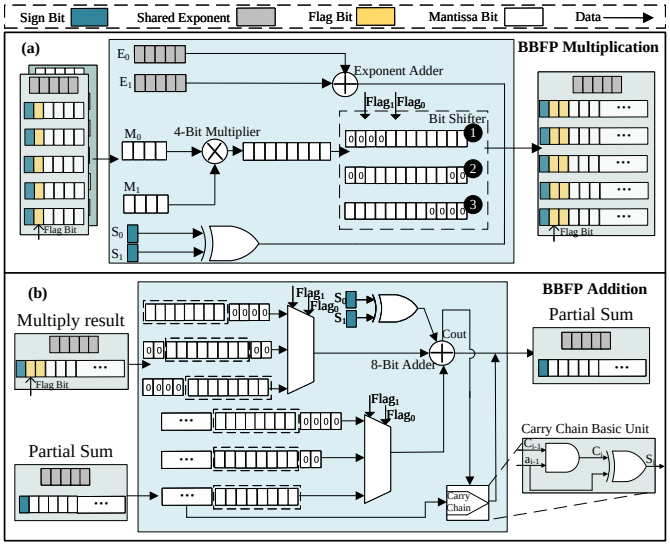


Fig. 5: (a) Multiplication operation between two BBFP blocks; (b) partial sum operation with BBFP and carry chain module.

a 4-bit multiplier, followed by a shift operation to obtain a 12-bit mantissa, as shown in Fig. 5 (a), where four bits are constant zero. To improve memory density, these zero bits are removed, and a 2-bit flag is used to represent the zero elements. For example, as shown in Fig. 5 (a), a flag of 00 corresponds to ❶ , while 01 or 10 corresponds to ❷ and 11 corresponds to ❸. So, the final output is a BBFP consisting of a 2-bit flag, 1-bit sign, and 8-bit mantissa.

Partial-Sum Addition: In the MAC module, after performing inter-block multiplication, partial-sum addition is required. We observed that the resulting data blocks from inter-block multiplication exhibit a regular bit-level sparsity pattern. Therefore, we employ sparse addition to reduce the adder bit-width and decrease resource consumption. Fig. 5 (b) illustrates three types of typical sparse adders. An 8-bit adder combined with a n-bit carry chain is used to replace a $12+n$-bit adder. The full adder expression is as follows:

$$S = CI \oplus a_i \oplus b_i \tag{11}$$

$$C = a_i b_i + C_i(a_i \oplus b_i) \tag{12}$$

Where a_i and b_i represent the i-th bit of the partial sum and the multiplication result, respectively. As shown in Fig. 5 (b), a_i is not always zero, whereas b_i becomes zero under specific patterns. This allows the adder to be simplified as follows:

$$S_c = C_i \oplus a_i \tag{13}$$

$$C_{co} = C_{ci} a_i \tag{14}$$

Comparing Eq. (14) and Eq. (13) with Eq. (12) and Eq. (11), the carry chain module reduces one AND gate and two XOR gates compared to the full adder module. For example by replacing the 12-bit adder with an 8-bit adder and a 4-bit carry chain, the adder unit achieves a 15% reduction in resource consumption. Furthermore, as the BBFP bit-width increases and the number of overlapping bits decreases, the optimization effect becomes increasingly pronounced.

MAC Efficiency: Table. I presents the MAC unit area and memory efficiency across different formats. Compared to BFP, the area for BBFP increases due to the separation of high and low mantissa groups. This leads to wider widths for both multiplication and partial-sum addition, resulting in higher area consumption. Additionally, since BBFP introduces an extra 1-bit flag bit, its memory efficiency

TABLE I: Various data types for MAC unit Mem Eff. and area.

Datatype	BlockSize	Area	Equivalent Bit-Width	Mem Eff.
FP 16	1	39599	16	1×
INT 8	1	9257	8	2×
BFP 8	32	9371	9.16	1.75×
BFP 6	32	5633	7.16	2.24×
BBFP(8,4)	32	9806	10.16	1.58×
BBFP(6,3)	32	5764	8.16	1.96×

is slightly lower. However, BBFP(6,3) offers the higher representation capability than BFP8, while consuming less area and memory footprint. This demonstrates that our new format provides stronger representational power and lower computational overhead.

B. Proposed nonlinear computing unit

Segment Lookup Table: Unlike FP32, where each data value has an exponent field, BBFP assigns a shared exponent field for a set of data. Therefore, we propose a segmented approach to load LUT, based on the shared exponent and perform lookups according to mantissas. Firstly, we divide the function values into several segments, based on different exponents, and store these segments in external memory. For instance, with five exponent bits, the function is split into $2^5 \times 2$ sub-tables. Secondly, once a shared exponent is calculated during the alignment phase, the corresponding sub-table can be loaded. Finally, unlike floating point LUT, which require additional mapping, BBFP uses the mantissa directly as the address for lookup.

Pipelined Design: To improve hardware throughput and mask the time required for loading LUT from external memory, the entire nonlinear computation unit is designed with a pipelined architecture. Each module is equipped with a buffer. Additionally, the computation unit is capable of computing different transcendental functions. However, different functions may require different computation sequences and computation units. To address this, the computation unit features adjustable computation order and is equipped with redundant units.

To illustrate the data flow adjustment for computing the sigmoid function, the formula for the sigmoid function is as follows:

$$Sigmoid(x) = 1/(1+e^{-x}) \tag{15}$$

First, the values of $(1+e^{-x})$ are computed offline, and the results are stored in external memory. The Control Unit then configures the data flow through the pipeline, directing it from the Align Exponent Unit to the LUT File, followed by the Div Unit, and finally to the Output Encoder. Similarly, the nonlinear computation unit can also compute functions such as SILU and GELU.

INT Computation: After the data passes through the Align Exponent module, it is converted from FP16 to BBFP for lookup table computation. By adjusting the data format in the table offline, each entry in the sub-table can be converted from FP16 to BBFP. This ensures that after the data passes through the LUT file, it retains the BBFP format for the next computation step.

Basic Unit: Based on the methods outlined above, we propose the nonlinear unit shown in Fig. 6. This nonlinear unit primarily consists of the Align Exponent Unit, Sub Unit, Mul Unit, Adder Tree, Div Unit, and LUT File. Fig. 6 illustrates the flow of the nonlinear unit using the softmax computation as an example.

C. LLMs Accelerator

Overall Architecture: The BBAL is shown in Fig. 7, which is centred around a weight stationary PE array and proposed nonlinear computation unit, accompanied by an input encoder, input buffer, weight buffer, output buffer, output encoder, max unit, FP encoder, FP adder, and control unit. Performing multiplication between two BFP requires only a single exponent addition. Therefore, the PE

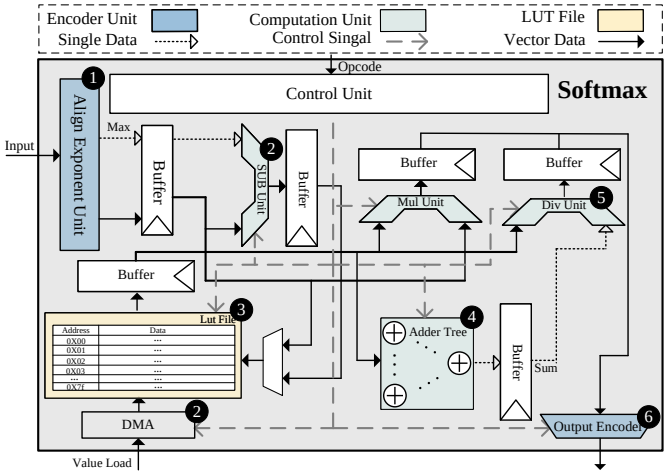


Fig. 6: Proposed nonlinear unit with Softmax as an example and the numbers in the figure indicate the computation sequence.

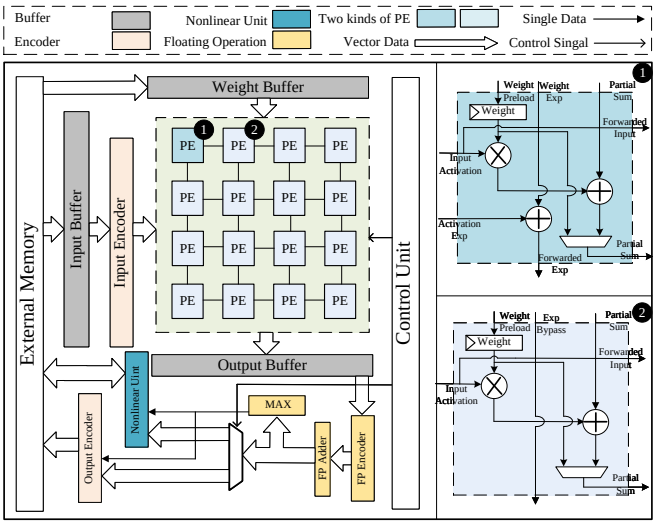


Fig. 7: A computation accelerator based on BBFP, incorporating two types of PE optimized for BBFP operations.

computation units have been partially modified to accommodate block floating point operations, resulting in two versions: ❶ adds a shared exponent adder, and ❷ includes a shared exponent bypass unit.

Computation Flow: When processing matrix multiplication, each 4×4 elements are encoded into BBFP and is sent to the PE array for computation. After computation, the data flows through the FP encoder, where it is encoded into FP format and sent to the FP adder, waiting to perform addition. After executing the addition, the data flows through a max unit and a data selector. The value output by the max unit can be used by either the nonlinear unit or the output encoder, eliminating the need for an additional comparator. Finally, the control unit determines whether the output from the floating point adder will pass through the nonlinear computation unit.

V. Evaluation

A. Configuration

Nonlinear Configuration: We used BBFP(10,5) to quantise the nonlinear layers, with the address width of each LUT being 7-bit. Considering the tendency of Softmax values towards zero, we designed 18 sub-tables, with SILU having 24 sub-tables. We replaced the nonlinear layers in the Llama-7B [5], Llama2-7B [34], and Llama3-8B [35] and evaluated their performance on the WikiText2 [36]. To validate the accuracy of our nonlinear quantisation strategy, we used FP32-based nonlinear units as the baseline and compared them with BBFP(10,5) and BFP10. Furthermore, to assess the efficiency of our quantisation strategy, we compared it with sota methods from three perspectives: EDP, ADP, and Efficiency (Throughput/(Area×Power)).

Linear Configuration: We modified the BBFP implementations to evaluate their impact on LLMs. The dataset used in this experiment was WikiText2. To validate the accuracy of the quantisation methods, we used FP16 as the baseline and compared BBFP with mantissa widths of 3, 4, and 6 bits against BFP4, BFP6, and sota methods, including OmniQuant [37], Oltron [38], and Olive [39], all of which quantise both weights and activations. To assess the efficiency of the quantisation strategy, we made comparisons with other approaches across three dimensions: area, energy, and throughput.

Implementation: We implemented the design with BBFP and BFP using Chisel [40] and evaluated it under the TSMC 28nm process. We utilized Design Compiler [41], and used CACTI [42] to estimate the area and power of on-chip memories. We developed a cycle-level simulator based on DnnWeaver [43] to estimate the performance.

B. Results Analysis:

Nonlinear Accuracy Analysis: Table IV presents a comparison of PPL between BBFP(10,5) and BFP10. Experimental results demonstrate that our nonlinear quantisation strategy incurs a maximum PPL increase of only 0.44 across the three models, whereas BFP10 results in at least 3× PPL increases. This demonstrates that BBFP(10,5) bridges the gap between BFP and application to nonlinear operations.

Nonlinear Efficiency Analysis: Table V compares our proposed nonlinear unit with other sota. Compared to low-precision approximation algorithms [32], our design shows a less favourable performance across the ADP and EDP. This is because our proposed nonlinear computation unit for LLMs requires full-precision, high-bitwidth integer multipliers and dividers to minimize numerical error, which increases both area and power consumption. Additionally, to enhance compatibility, several redundant units are included; for example, the vector multiplication module remains idle during softmax computation, further contributing to larger area requirements and increased static power consumption. However, leveraging a segmented-exponent dynamic lookup strategy allows the design to reduce costly on-chip memory by utilizing more affordable off-chip memory, achieving high compatibility and efficiency. Additionally, because BBFP preserves the computational efficiency of fixed point, it results in significant conservation of computational resources. Experimental results demonstrate that our design achieves nearly a 30× efficiency improvement over high-precision method [33].

Linear Area Analysis: Table III presents the area of a PE under various methods. The PE area consists of two components: multiplier and adder, with multiplier occupying the majority. Hence, BFP, Olive, Oltron and BBFP, with the same mantissa width, exhibit similar areas.

Linear Accuracy-Throughput Analysis: Table II shows the effects of BBFP quantisation for the linear layers without any calibration. Experimental results show that BBFP offers improved accuracy compared to BFP. Specifically, BBFP(3,1) achieves a 6% improvement over BFP4, and BBFP(4,2) achieves an average PPL only 4% higher than BFP6. Additionally, BBFP(4,2) achieve 30% lower PPL, compared to Oltron, and 33%, compared to OmniQuant.

Through analyzing activation distribution across different models, we observe that models contain varying proportions and magnitudes of outliers. Hence, outlier-aware quantisation methods, which capture a fixed proportion of outliers, perform poorly on the Llama (with more outliers) but achieve better results on the OPT (with fewer

TABLE II: Perplexity results of quantized model on Wikitext2 (lower is better).

Model	Llama-1B	Llama-3B	Llama-7B	Llama-13B	Llama-30B	Llama-65B	OPT-1.3B	OPT-2.7B	OPT-6.7B	OPT-13B	OPT-30B	OPT-66B
FP16	9.88	7.87	5.47	5.09	4.10	3.53	14.62	12.47	10.86	10.12	9.56	9.34
Oltron	N/A	N/A	14.67	9.48	7.51	6.69	N/A	N/A	11.99	11.65	10.60	10.29
Olive	N/A	N/A	144.78	42.24	36.55	NaN	N/A	N/A	107.15	416.57	334.7	4058.83
Omniquant	N/A	N/A	11.26	10.87	10.33	9.17	N/A	N/A	12.24	11.65	10.6	10.29
BFP6(6m)	10.06	7.95	5.61	5.13	4.12	3.61	15.57	12.5	10.91	10.22	9.62	9.48
BFP4(4m)	13.45	9.44	5.83	5.72	5.05	4.12	27.21	18.98	12.24	11.56	10.50	10.10
BBFP(3,1)	12.35	9.00	5.66	5.33	4.46	4.01	23.12	15.29	14.07	10.85	10.45	10.27
BBFP(4,2)	10.41	8.13	5.80	5.39	4.37	3.65	17.06	13.36	12.03	10.39	9.63	9.87
BBFP(4,3)	10.65	8.20	5.80	5.20	4.26	3.69	17.52	13.89	11.54	10.38	9.61	9.93
BBFP(6,3)	**9.93**	**7.89**	**5.48**	**5.09**	**4.10**	**3.59**	**15.16**	**12.49**	**10.89**	**10.12**	**9.55**	9.38
BBFP(6,4)	9.93	7.9	5.48	5.09	4.10	3.59	15.00	12.47	10.89	10.14	9.55	**9.36**

TABLE III: Comparison of PE area (μm^2) across different quantization strategies (normalized by maximum BBFP(6,3) PE area).

	Oltron	Olive	BFP4	BFP6	BBFP(3,1)	BBFP(3,2)	BBFP(4,2)	BBFP(4,3)	BBFP(6,3)	BBFP(6,4)	BBFP(6,5)
Area	78.50	156.47	215.23	110.24	77.69	**75.51**	117.11	113.31	241.01	231.14	224.70
Norm	0.33	0.65	0.46	0.90	0.32	0.31	0.49	0.47	1.00	0.96	0.93

TABLE IV: PPL of Llama at various schemes with nonliear units.

Data Format	nonlinear Operation	LLama-7B	Llama2-7B	Llama3-8B
FP32	Altogether	5.68	5.47	6.14
BBFP(10,5)	Softmax Only	5.74	5.62	6.24
	SILU Only	5.71	5.53	6.21
	Altogether	5.81	5.91	6.34
BFP10	Softmax Only	67.31	32.72	69.95
	SILU Only	33.21	17.54	31.30
	Altogether	99.28	50.21	102.35

TABLE V: Comparison of ADP, EDP, Eff., and Compatibility.

Methods	**Num**	**Format**	**ADP↓**	**EDP↓**	**Eff.↑**	**Compatibility**
[32]	10	Int8	∼4.33	∼79.58	∼85.98	-
[33]	8	Int 27	∼299.13	∼18691.24	3.31	-
Ours	16	BBFP(10,5,5)	32.64	1040.40	98.03	SILU and so on

outliers), as shown in Fig. 8. BBFP's ability to capture any proportion of outliers ensures a stable accuracy baseline across models.

Fig. 8 presents the accuracy and throughput performance of various quantisation strategies under the same PE iso-area condition. Since BBFP(3,1), BBFP(3,2), and Oltron all use 3-bit multipliers and low-bit adders, they exhibit similar throughput within the same area. However, due to BBFP's superior outlier protection capability, BBFP(3,1) achieves a 22% accuracy improvement on average across all tasks compared to Oltron. Additionally, compared to BFP4, BBFP(3,1) and BBFP(3,2) achieve a 40% throughput improvement while maintaining similar accuracy. The BBFP with a width of 4 shows a 30% drop in throughput compared to Oltron, but its PPL is reduced by 30%.

Energy Analysis: Fig. 9 illustrates the average energy consumption for various methods when executing tasks under the same number of PE and with the same buffer size. Compared to BFP4, BBFP, with a width of 3, reduces energy consumption by 13%, primarily attributed to reductions in both static and core energy. The energy consumption increase of BBFP compared to BFP with the same mantissa bit-width is within 5%. This increase is primarily due to the larger area, which results in higher static and dynamic power, as well as the additional 1-bit flag in BBFP contributing to slightly higher DRAM power.

VI. Conclusion

In this paper, we propose BBFP, which is optimized based on BFP, to reduce quantisation error and promote the further application of BFP. The key insight is to reduce the probability of aligning maximum exponent as the shared exponent, thereby capturing outliers while minimizing quantisation error. Due to the low quantisation

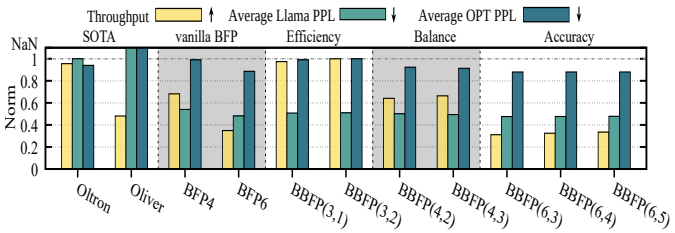


Fig. 8: Various quantisation methods compared by average Llama/OPT PPL (lower is better) and throughput for equal PE area.

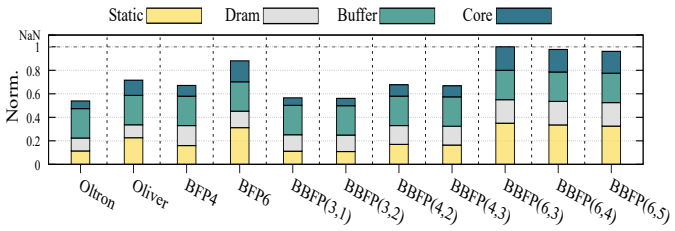


Fig. 9: Average normalized energy consumption under identical PE count and buffer size (excluding nonlinear unit).

error of BBFP, we propose an efficient nonlinear computation unit based on BBFP, and this extends the application potential of BFP. Experimental results demonstrate that proposed nonlinear computation unit achieves a 30× efficiency improvement over high-precision nonlinear computation unit, with almost no accuracy loss. Finally, we propose BBAL, primarily comprising an optimized PE array based on BBFP, paired with proposed nonlinear computation unit. Experimental results demonstrate that BBAL achieves a 22% accuracy improvement compared to outlier-aware accelerators with similar hardware consumption, and a 40% throughput improvement compared to BFP4 with similar accuracy.

VII. Acknowledgement

We appreciate the reviewers for their helpful feedback. This work is supported by the National Key Research and Development Program (Grant No. 2024YFB4405600), the National Natural Science Foundation of China (Grant No. 62472086), the Basic Research Program of Jiangsu (Grant No. BK20243042), the Science and Technology Major Special Program of Jiangsu (No. BG2024010), and the Start-up Research Fund of Southeast University (Grant No. RF1028624005).

REFERENCES

[1] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, vol. 1, 2020.

[2] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," *Advances in neural information processing systems*, vol. 35, pp. 36 479–36 494, 2022.

[3] J. Kil, S. Changpinyo, X. Chen, H. Hu, S. Goodman, W.-L. Chao, and R. Soricut, "Prestu: Pre-training for scene-text understanding," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 15 270–15 280.

[4] C. Lyu, J. Xu, and L. Wang, "New trends in machine translation using large language models: Case examples with chatgpt," *arXiv preprint arXiv:2305.01181*, 2023.

[5] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[6] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024.

[7] N. Koilia and C. Kachris, "Hardware acceleration of llms: A comprehensive survey and comparison," *arXiv preprint arXiv:2409.03384*, 2024.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[9] W. Huang, Y. Liu, H. Qin, Y. Li, S. Zhang, X. Liu, M. Magno, and X. Qi, "Billm: Pushing the limit of post-training quantization for llms," *arXiv preprint arXiv:2402.04291*, 2024.

[10] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 318–30 332, 2022.

[11] Z. Yuan, L. Niu, J. Liu, W. Liu, X. Wang, Y. Shang, G. Sun, Q. Wu, J. Wu, and B. Wu, "Rptq: Reorder-based post-training quantization for large language models," *arXiv preprint arXiv:2304.01089*, 2023.

[12] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[13] N. Burgess, J. Milanovic, N. Stephens, K. Monachopoulos, and D. Mansell, "Bfloat16 processing for neural networks," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 88–91.

[14] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022.

[15] C. Zhang, J. Cheng, I. Shumailov, G. A. Constantinides, and Y. Zhao, "Revisiting block-based quantisation: What is important for sub-8-bit llm inference?" *arXiv preprint arXiv:2310.05079*, 2023.

[16] T. Yeh, M. Sterner, Z. Lai, B. Chuang, and A. Ihler, "Be like water: Adaptive floating point for machine learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 25 490–25 500.

[17] S. Fox, S. Rasoulinezhad, J. Faraone, P. Leong *et al.*, "A block minifloat representation for training deep neural networks," in *International Conference on Learning Representations*, 2020.

[18] L. Zou, W. Zhao, S. Yin, C. Bai, Q. Sun, and B. Yu, "Bie: Bi-exponent block floating-point for large language models quantization," in *Forty-first International Conference on Machine Learning*, 2024.

[19] M. Cho and Y. Kim, "Fpga-based convolutional neural network accelerator with resource-optimized approximate multiply-accumulate unit," *Electronics*, vol. 10, no. 22, p. 2859, 2021.

[20] H. Fan, S. Liu, Z. Que, X. Niu, and W. Luk, "High-performance acceleration of 2-d and 3-d cnns on fpgas using static block floating point," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 4473–4487, 2021.

[21] Z. Song, Z. Liu, and D. Wang, "Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need. advances in neural information processing systems," *Advances in neural information processing systems*, vol. 30, no. 2017, 2017.

[23] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 469–474.

[24] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[25] S.-y. Liu, Z. Liu, X. Huang, P. Dong, and K.-T. Cheng, "Llm-fp4: 4-bit floating-point quantized transformers," *arXiv preprint arXiv:2310.16836*, 2023.

[26] X. Hu, Y. Cheng, D. Yang, Z. Yuan, J. Yu, C. Xu, and S. Zhou, "I-llm: Efficient integer-only inference for fully-quantized low-bit large language models," *arXiv preprint arXiv:2405.17849*, 2024.

[27] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[28] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[29] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, "Post-training quantization for vision transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 092–28 103, 2021.

[30] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8bert: Quantized 8bit bert," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 2019, pp. 36–39.

[31] K. Kalliojarvi and J. Astola, "Roundoff errors in block-floating-point systems," *IEEE transactions on signal processing*, vol. 44, no. 4, pp. 783–790, 1996.

[32] G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, A. Nannarelli, M. Re, and S. Spanò, "A pseudo-softmax function for hardware-based high speed image classification," *Scientific reports*, vol. 11, no. 1, p. 15307, 2021.

[33] Y. Zhang, L. Peng, L. Quan, Y. Zhang, S. Zheng, and H. Chen, "High-precision method and architecture for base-2 softmax function in dnn training," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 8, pp. 3268–3279, 2023.

[34] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[35] Meta, "Meta llama 3: Advancing generative ai responsibly," https://ai.meta.com/blog/meta-llama-3/, 2024.

[36] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[37] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, "Omniquant: Omnidirectionally calibrated quantization for large language models," *arXiv preprint arXiv:2308.13137*, 2023.

[38] C. Xue, C. Zhang, X. Jiang, Z. Gao, Y. Lin, and G. Sun, "Oltron: Algorithm-hardware co-design for outlier-aware quantization of llms with inter-/intra-layer adaptation," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[39] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[40] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[41] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys®*. Springer Science & Business Media, 1997.

[42] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[43] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J. K. Kim, A. Mishra, and H. Esmaeilzadeh, "Dnnweaver: From high-level deep network models to fpga acceleration," in *the Workshop on Cognitive Architectures*, 2016.

CirSTAG: Circuit Stability Analysis on Graph-based Manifolds

Wuxinlin Cheng
Stevens Institute of Technology
wcheng7@stevens.edu

Yihang Yuan
Stevens Institute of Technology
yyuan22@stevens.edu

Chenhui Deng
NVIDIA Research
cdeng@nvidia.com

Ali Aghdaei
University of California, San Diego
aaghdaei@ucsd.edu

Zhiru Zhang
Cornell University
zhiruz@cornell.edu

Zhuo Feng
Stevens Institute of Technology
zfeng12@stevens.edu

***Abstract*—Circuit stability (sensitivity) analysis aims to estimate the overall performance impact of variations in underlying design parameters, such as gate sizes and capacitance. This process is challenging because it often requires numerous time-consuming circuit simulations. In contrast, graph neural networks (GNNs) have shown remarkable effectiveness and efficiency in tackling several chip design automation issues, including circuit timing predictions, parasitic prediction, gate sizing, and device placement. This paper introduces a novel approach called CirSTAG, which utilizes GNNs to analyze the stability (robustness) of modern integrated circuits (ICs). CirSTAG is grounded in a spectral framework that examines the stability of GNNs by leveraging input/output graph-based manifolds. When two adjacent nodes on the input manifold are mapped (through a GNN model) to two remote nodes (data samples) on the output manifold, this indicates a significant mapping distortion (DMD) and consequently poor GNN stability. CirSTAG calculates a stability score equivalent to the local Lipschitz constant for each node and edge, considering both graph structure and node feature perturbations. This enables the identification of the most critical (sensitive) circuit elements that could significantly impact circuit performance. Our empirical evaluations across various timing prediction tasks with realistic circuit designs demonstrate that CirSTAG can accurately estimate the stability of each circuit element under diverse parameter variations. This offers a scalable method for assessing the stability of large integrated circuit designs.**

I. Introduction

Stability analysis is crucial in the design and optimization of Very Large Scale Integration (VLSI) circuits, ensuring reliable performance under perturbations such as design parameter changes (e.g., transistor sizing), process and temperature variations, and supply voltage fluctuations [1]. Assessing circuit stability is essential to prevent unintended behaviors like oscillations or unpredictable outputs, which can adversely impact the functionality and reliability of integrated circuits (ICs). Moreover, stability analysis guides circuit optimization tasks, such as gate sizing for timing and power optimization [21], by identifying the most unstable circuit nodes that, when modified, can significantly improve overall performance. However, existing stability analysis methods often require numerous repeated circuit simulations after perturbing underlying parameters, becoming prohibitively expensive for large-scale designs.

Graph Neural Networks (GNNs) have emerged as powerful tools in machine learning, particularly for graph-structured data [26], [32]. By integrating graph structures and node features, GNNs produce low-dimensional embedding vectors that preserve graph structural information [16]. They have been successfully applied in various real-world applications, including recommendation systems [14], traffic flow prediction [29], device placement [23], and VLSI timing prediction [17].

In this work, we present **CirSTAG**, a novel framework to quantify the stability of circuit networks leveraging GNNs. For a pre-trained GNN model that mimics the behaviors of a family of circuit designs (e.g., pre-routing slack prediction [17]), CirSTAG assesses circuit stability by estimating the impact of potential input perturbations (e.g., graph topology and node feature modifications) on the GNN's output. Specifically, if two nearby circuit nodes on the input graph are mapped to distant embedding vectors by the GNN, it indicates a substantial *distance mapping distortion* (DMD) and poor circuit stability. To address this, we propose an efficient spectral framework for quantifying DMDs of GNNs on input/output graph-based manifolds. We also introduce a stability score for each node (or edge) computed based on the DMD estimation, which is equivalent to the local Lipschitz constant of the GNN model.

CirSTAG enjoys near-linear runtime complexity and is compatible with various GNN architectures due to its data-centric nature. Experimental results demonstrate CirSTAG's capability to gauge individual node stability within diverse GNN models for realistic circuit designs.

Our key contributions are summarized as follows:

- To the best of our knowledge, we introduce the first spectral framework for analyzing circuit stability by evaluating node-level stability in GNNs, achieved by measuring the DMD of adjacent nodes using input/output graph-based manifolds.
- We construct proper graph-based manifolds for estimating DMDs by exploiting probabilistic graphical models (PGMs) and spectral graph embedding, preserving key structural properties of the given graph dataset.

- CirSTAG has near-linear time complexity and is agnostic to GNN models and node label information. Our empirical results validate that our approach provides a scalable method for assessing individual node (gate) stability in pre-routing timing prediction [17] and reverse engineering of realistic circuit designs [4].

The rest of the paper is organized as follows. In Section II, we provide background on concepts related to DMD and PGM. Section III offers an overview of the proposed CirSTAG framework. In Section IV, we introduce the technical details of CirSTAG, including its algorithm flow and complexity. In Section V, we present experimental results evaluating the performance of CirSTAG on stability analysis tasks using real-world VLSI design benchmarks. Finally, we conclude the paper in Section VI.

II. Background

A. Applications of GNNs in Electronic Design Automation

GNNs have demonstrated significant potential in Electronic Design Automation (EDA), revolutionizing tasks ranging from circuit design to verification by modeling complex relationships within electronic circuits.

In circuit analysis, GNNs enable efficient modeling and simulation of large-scale circuits, leading to accurate predictions of circuit behavior under various conditions [3]. For optimization, they facilitate the identification of optimal circuit configurations [24]. GNNs also aid in fault detection and diagnosis by pinpointing potential faults within electronic systems and suggesting corrective measures, thus improving reliability and reducing maintenance costs [7].

In layout and routing, GNNs manage the complexity of modern integrated circuits by optimizing layouts for better performance and lower power consumption while ensuring fabrication constraints are met [23]. Additionally, they have been applied to timing analysis, predicting arrival times and slack at timing endpoints, enhancing the efficiency and accuracy of EDA processes [17].

Furthermore, GNNs have emerged as transformative tools in circuit reverse engineering and security. By deciphering relationships between circuit components, GNNs facilitate the extraction and reconstruction of circuit functionalities from physical layouts. This capability is crucial for understanding proprietary designs, detecting unauthorized alterations, and ensuring compliance with intellectual property laws. Recent advancements demonstrate that GNNs can accurately identify sub-circuits and their functionalities, advancing the field of hardware security and trust [4], [28].

B. Manifold-Based Stability Analysis of ML Models

The stability of the ML model refers to its ability to produce consistent outputs despite small variations or noise in the input [25]. Let F denote an ML model that operates on input X to yield output Y, i.e., $Y = F(X)$. The stability of F can be evaluated by measuring distortions in the mapping between low-dimensional input and output manifolds. Specifically, the *distance mapping distortion* (DMD) metric quantifies distance

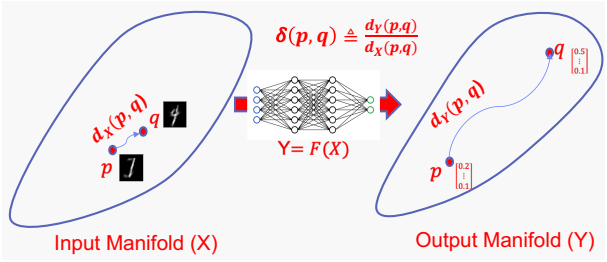


Fig. 1: Distance mapping distortions on manifolds [8]

distortions between graph-based manifolds [8], as illustrated in Fig. 1. For two input data samples p and q, the DMD metric $\delta^F(p,q)$ is defined as the ratio of the distance $d_Y(p,q)$ on the output manifold $G_Y = (V, E_Y)$ to the distance $d_X(p,q)$ on the input manifold $G_X = (V, E_X)$:

$$\delta^F(p,q) \stackrel{\text{def}}{=} \frac{d_Y(p,q)}{d_X(p,q)}. \tag{1}$$

Evaluating DMD between pairs of data samples helps determine the stability of the ML model. If two data samples that are close on the input manifold are mapped to distant points on the output manifold (i.e., $\delta^F(p,q)$ is large), this indicates a large local Lipschitz constant and poor stability of the model near those samples.

C. Graph Topology Learning via PGM

Given M samples of N-dimensional vectors stored in a data matrix $X \in \mathbb{R}^{N\times M}$, recent graph topology learning methods [11]–[13] estimate the graph structure (specifically, the Laplacian matrix L) from X to achieve two main objectives: **(1) Signal Smoothness over the Graph:** This can be quantified by the matrix trace $\mathrm{Tr}(X^\top L X)$ [18]. **(2) Sparsity of the Estimated Graph Topology:** Encouraging sparsity leads to simpler and more interpretable graph structures. Consider a random vector $x \sim \mathcal{N}(0, \Sigma)$ with probability density function:

$$f(x) = \frac{\exp\left(-\frac{1}{2}x^\top \Sigma^{-1} x\right)}{(2\pi)^{N/2}\det(\Sigma)^{1/2}} \propto \det(\Theta)^{1/2}\exp\left(-\frac{1}{2}x^\top \Theta x\right), \tag{2}$$

where $\Sigma = \mathbb{E}[xx^\top] \succ 0$ is the covariance matrix, and $\Theta = \Sigma^{-1}$ is the precision matrix (inverse covariance matrix). When the sample covariance matrix S is obtained from M i.i.d. samples $X = [x_1, \ldots, x_M]$ drawn from $\mathcal{N}(0, \Sigma)$, each element $\Theta_{i,j}$ of the precision matrix encodes the conditional dependence between variables X_i and X_j. Specifically, $\Theta_{i,j} = 0$ implies that variables X_i and X_j are conditionally independent, given the rest.

The maximum likelihood estimation (MLE) of the precision matrix Θ (i.e., learning the Probabilistic Graphical Model) can be formulated as the following convex optimization problem [12], [19]:

$$\max_{\Theta} \quad F(\Theta) = \log\det(\Theta) - \frac{1}{M}\mathrm{Tr}(X^\top \Theta X), \tag{3}$$

subject to $\Theta = L + \frac{1}{\sigma^2}I$, where L is a valid graph Laplacian matrix, I is the identity matrix, and $\sigma^2 > 0$ is a prior feature variance.

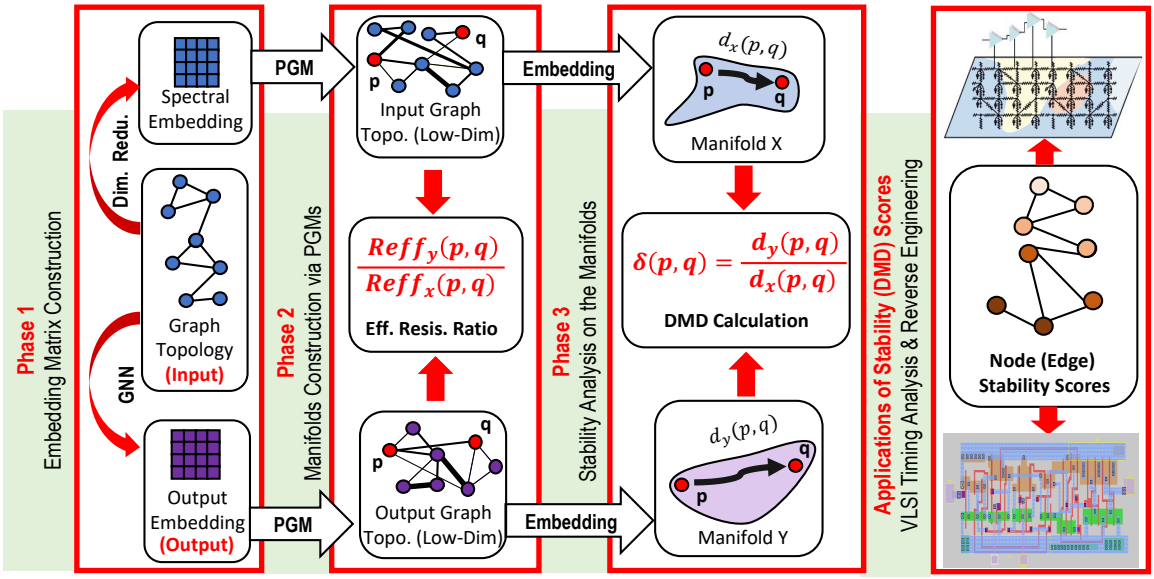


Fig. 2: The proposed GNN-based circuit stability analysis framework (CirSTAG) and its applications.

III. Overview of CirSTAG

To quantify the stability of an ML model, the DMD metric has been introduced [8] to provide an effective measure of how an ML model distorts pairwise distances between data samples. However, applying DMD to GNNs presents several challenges. DMD assumes that both input and output data can be well-represented by low-dimensional graph-based manifolds [8]. While this assumption typically holds for GNN outputs (i.e., node embedding vectors), it often does not for input circuit graphs, which may reside in relatively high-dimensional spaces. As shown in Fig. 4, directly using the original circuit networks (graphs) as the input manifold fails to accurately predict circuit delay stability under node feature (e.g., capacitance) perturbations.

Fig. 2 presents an overview of the proposed **CirSTAG** framework and its applications in design automation tasks. Our methodology consists of three distinct phases:

1) **Phase 1**: We construct an input embedding matrix by employing spectral graph embedding to retain the original graph's structural characteristics. We also obtain an output embedding matrix generated by the GNN model.
2) **Phase 2**: We create low-dimensional input and output graph-based manifolds using a highly scalable PGM construction algorithm.
3) **Phase 3**: We quantify node stability by computing DMDs that encapsulate the mappings (via GNNs) between the input and output graph-based manifolds.

In the following sections, we delve into the specifics of our CirSTAG framework.

IV. Methodology

A. Phase 1: Input/Output Embedding Matrices Construction

a) Nonlinear Dimensionality Reduction of the Input Graph: To reduce the dimensionality of the input graph for manifold construction, we employ a nonlinear dimensionality reduction approach inspired by the *Laplacian Eigenmap* algorithm [5]. The key idea is to construct a graph-based manifold from the high-dimensional data and then map each data sample (node) into a low-dimensional representation using the first few Laplacian eigenvectors. Specifically, given a graph with adjacency matrix A and degree matrix D, we compute the normalized graph Laplacian matrix $L_{\text{norm}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where I is the identity matrix. We then obtain the top M smallest eigenvalues $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_M$ and their corresponding eigenvectors $\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_M$ of L_{norm}. We construct the input spectral embedding matrix to encapsulate key structural characteristics of the graph [10]:

$$U_M = \left[\sqrt{|1 - \tilde{\lambda}_1|}\, \tilde{u}_1, \ldots, \sqrt{|1 - \tilde{\lambda}_M|}\, \tilde{u}_M \right]. \quad (4)$$

This embedding matrix U_M effectively reduces the dimensionality of the input graph while preserving its essential structural properties. In the next phase, we will use U_M to construct the input graph-based manifold via graph topology learning.

b) Output Embedding from GNNs: Since the outputs of GNNs are typically low-dimensional node embeddings, we directly use the node embeddings generated by the GNN as the output embedding matrix. This output embedding matrix will similarly be transformed into an output graph-based manifold through graph topology learning in the subsequent phase.

B. Phase 2: Constructing Graph-Based Manifolds via PGMs

In this phase, we construct the input and output graph-based manifolds using the embedding matrices obtained in Phase 1. This process is motivated by a recent PGM-based graph topology learning framework (SGL) [15], [30]. However, the original SGL framework requires numerous iterations to converge, leading to superlinear runtime complexity. To enhance scalability, we introduce an efficient spectral sparsification scheme for learning PGMs. Consider a weighted undirected

graph $G = (V, E, w)$, with edge weights $w \in \mathbb{R}^V_{\geq 0}$ and $|V| = N$. The Laplacian matrix L of G can be expressed as:

$$L = \sum_{(p,q)\in E} w_{p,q} e_{p,q} e_{p,q}^\top, \tag{5}$$

where $e_{p,q} = e_p - e_q$, $w_{p,q}$ is the weight of edge (p, q), and $e_p \in \mathbb{R}^N$ is the standard basis vector with a 1 at the p-th entry. We aim to solve the following convex optimization problem [12]:

$$\begin{aligned} &\max_{\Theta} : F(\Theta) = F_1 - \frac{1}{M} F_2 \\ &F_1 = \log\det(\Theta) = \sum_{i=1}^{N} \log(\lambda_i + \frac{1}{\sigma^2}) \\ &F_2 = Tr(X^\top \Theta X) = \frac{Tr(X^\top X)}{\sigma^2} + \sum_{(p,q)\in E} w_{p,q} \|X^\top e_{p,q}\|_2^2 \end{aligned} \tag{6}$$

where $\Theta = L + \frac{1}{\sigma^2} I$, λ_i are the Laplacian eigenvalues, and $X \in \mathbb{R}^{N\times M}$ is the input data matrix equal to the spectral embedding matrix U_M from (4). To maximize $F(\Theta)$, we compute the partial derivatives with respect to $w_{p,q}$ [30]:

$$\begin{aligned} \frac{\partial F_1}{\partial w_{p,q}} &= R^{\text{eff}}_{p,q}, \\ \frac{\partial F_2}{\partial w_{p,q}} &= D^{\text{data}}_{p,q} = \|X^\top e_{p,q}\|_2^2 = \frac{1}{w_{p,q}}, \end{aligned} \tag{7}$$

where $R^{\text{eff}}_{p,q}$ is the effective resistance between nodes p and q, and $D^{\text{data}}_{p,q}$ is the ℓ_2 distance between their data vectors. Based on these gradients, we propose to prune non-critical edges from an initial dense graph by considering the spectral distortion metric [15]:

$$\eta_{p,q} = \frac{R^{\text{eff}}_{p,q}}{D^{\text{data}}_{p,q}} = w_{p,q} R^{\text{eff}}_{p,q}. \tag{8}$$

Edges with low effective resistance but large data distance (i.e., small $\eta_{p,q}$) are pruned. This strategy effectively maximizes F_2 without significantly decreasing F_1, allowing us to construct PGMs via spectral graph sparsification.

a) Initial Dense Graph Construction: Given the low-dimensional nature of the data from Phase 1, we efficiently construct the initial dense graph using the k-nearest neighbor (kNN) algorithm [22], which has computational complexity $O(|V| \log |V|)$.

b) PGM Refinement via Spectral Sparsification: To refine the PGM, we apply spectral graph sparsification using a short-cycle decomposition scheme [9]. This method partitions the graph into disjoint cycles by removing a fixed number of edges, ensuring cycles are bounded in length. Recent algorithms (Lemma 1) combine short-cycle decomposition with low-stretch spanning trees (LSSTs) to preserve the spectral properties of the original graph [20].

Lemma 1: Spectral sparsification of an unweighted, undirected graph G with Laplacian L_G can be achieved using a short-cycle decomposition algorithm, yielding a sparsified graph H with Laplacian L_H such that for all real vectors x, $x^\top L_G x \approx x^\top L_H x$ [9].

Extending prior work, we introduce a novel low-resistance-diameter (LRD) decomposition scheme to handle weighted graphs by restricting cycle lengths measured via effective resistance. Our method efficiently computes the effective resistance of each edge and uses a multilevel framework [2] to decompose the graph into cycles bounded by a given effective resistance threshold.

C. Phase 3: Stability Analysis on the Manifolds

To quantify the stability of the ML model, we utilize the DMD (as defined by equation (1)) [8], which measures how the model distorts pairwise distances between data samples. While geodesic distances are natural choices for d_X and d_Y, computing all-pairs geodesic distances is computationally prohibitive for large graphs [27]. To address this, we adopt the *effective resistance distance* [6], which can be efficiently computed and captures global graph structure. Suppose the input X is mapped to output Y by $Y = F(X)$. Let L_X and L_Y denote the Laplacian matrices of the input and output manifolds G_X and G_Y, respectively. Recent theoretical results [8] allow us to assess node stability using the largest generalized eigenvalues and corresponding eigenvectors of $L_Y^+ L_X$. We compute the weighted eigensubspace matrix $V_s \in \mathbb{R}^{N\times s}$: $V_s = \left[v_1\sqrt{\zeta_1},\ \ldots,\ v_s\sqrt{\zeta_s}\right]$, where $\zeta_1 \geq \zeta_2 \geq \cdots \geq \zeta_s$ are the largest s eigenvalues of $L_Y^+ L_X$, and $v_1, v_2, \ldots, v_s$ are the corresponding eigenvectors. Using V_s, we embed the input graph G_X, associating each node with an s-dimensional vector. The stability of an edge $(p, q) \in E_X$ is quantified by the embedding distance between its end nodes: $\|V_s^\top e_{p,q}\|_2^2$, where $e_{p,q} = e_p - e_q$, and e_p is the standard basis vector. The stability score of a node p is estimated as:

$$\begin{aligned} \textbf{Score}^F(p) &\stackrel{\text{def}}{=} \frac{1}{|\mathbb{N}_X(p)|} \sum_{q_i \in \mathbb{N}_X(p)} \left(\|V_s^\top e_{p,q_i}\|_2^2\right) \\ &\propto \frac{1}{|\mathbb{N}_X(p)|} \sum_{q_i \in \mathbb{N}_X(p)} \left(\delta^F(p, q_i)\right)^3, \end{aligned} \tag{9}$$

where $\mathbb{N}_X(p)$ denotes the set of neighbors of node p in G_X. This node stability score serves as a surrogate for the local Lipschitz constant $\|\nabla F(p)\|$ under the manifold setting [8].

D. Runtime Complexity

For spectral graph embedding, we can exploit fast multilevel eigensolvers that allow computing the first c Laplacian eigenvectors in nearly-linear time $O(c|V|)$ without loss of accuracy [31]. Then, the k-nearest neighbor algorithm [22] computational complexity is $O(|V| \log |V|)$. Sparsification via short-cycle decomposition has $O(|V|dm)$ time complexity. By leveraging fast Laplacian solvers [31], all nodes' stability scores can be computed with $O(|E|)$, where the V/E denotes the number of nodes/edges, d is the average degree of the matrix, and m is the order of Krylov subspace. The proposed CirSTAG algorithm flow is shown in Algorithm 1.

Algorithm 1 The algorithm flow of CirSTAG

Input: Input graph G, output Y
Output: Node stability scores in (9).

1: $U_N \leftarrow$ compute_weighted_spectral_embedding(G)
2: $kNN_dense_graph \leftarrow$ construct_kNN_graph(U_N)
3: $input_PGM \leftarrow$ sparse_graph(kNN_dense_graph)
4: $kNN_dense_graph_Y \leftarrow$ construct_kNN_graph(Y)
5: $output_PGM \leftarrow$ sparse_graph$(kNN_dense_graph_Y)$
6: $L_X \leftarrow$ compute_laplacian$(input_PGM)$
7: $L_Y \leftarrow$ compute_laplacian$(output_PGM)$
8: $V_N \leftarrow$ calculate_generalized_eigenvectors(L_Y, L_X)
9: **for** each node p in G_X **do**
10: Compute its stability score by:

$$\textbf{Score}(p) = \frac{1}{|\mathbb{N}_X(p)|} \sum_{q_i \in \mathbb{N}_X(p)} \left(\|V_s^\top e_{p,q_i}\|_2^2 \right)$$

11: **end for**

V. Experimental Results

To demonstrate the capability of **CirSTAG** for stability analysis in EDA applications, we present two case studies using high-accuracy GNN models: **(A)** Circuit Timing Prediction [17] and **(B)** Circuit Functional Reverse Engineering [4]. The effectiveness of CirSTAG relies on accurate predictions from the GNN models, which serve as black-box simulators for their respective tasks. Accurate embeddings ensure that the computed DMDs reflect true circuit behavior, allowing CirSTAG to capture performance variations due to node feature or topology perturbations. In Case Study (A), the GNN predicts signal arrival times (delays) at each circuit node, focusing on **node feature perturbations**. In Case Study (B), we examine the impact of **topology perturbations** on the cosine similarity of embeddings and the F1 macro score for multi-task classification of sub-circuits. These studies validate our methodology by assessing node stability under different perturbations.

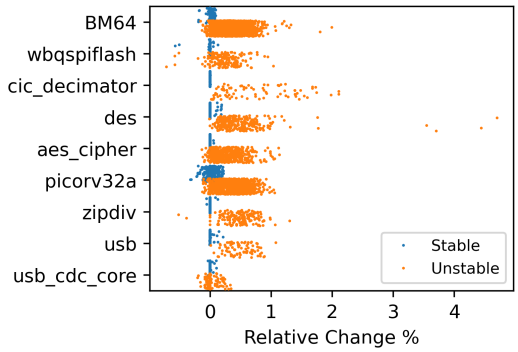


Fig. 3: Circuit delay variations (with dimension reduction).

A. Stability Analysis of Circuit Delays under Node Feature Perturbations

In this case study, we analyze circuit delay stability considering node feature (pin capacitance) perturbations using datasets and a GNN model from [17]. The GNN predicts pre-routing static timing analysis (STA) results, interpreting graphs where nodes represent cell pins and edges represent net connections and internal cell connections. We selected nine circuit datasets with the highest R^2 scores (96.88% to 99.22%) to ensure accurate GNN predictions. The GNN then predicted arrival times at primary output pins, and we computed the relative output change compared to unperturbed data. Nodes representing output pins were excluded, as they do not directly affect internal timing dynamics. Table I summarizes the relative changes in arrival time predictions. The results consistently show that perturbing unstable nodes leads to higher average and maximum relative changes compared to stable nodes, indicating greater sensitivity to capacitance changes. Increasing the scaling factor from 5 to 10 nearly doubles the relative change. However, increasing the percentage of perturbed nodes from 5% to 15% does not proportionally increase the relative change for unstable nodes, suggesting that the most unstable nodes contribute more significantly to variability. Fig. 3 illustrates the distribution of relative changes when perturbing the top 10% of nodes with a scaling factor of 10, highlighting the contrast between unstable and stable nodes.

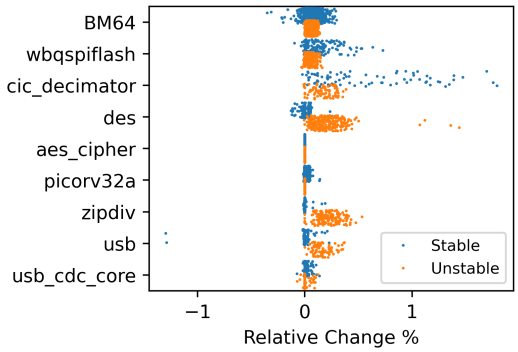


Fig. 4: Circuit delay variations(without dimension reduction).

Ablation Study of CirSTAG We conducted an ablation study by omitting the graph dimensionality reduction stage in CirSTAG. As shown in Fig. 4, the distribution of relative changes becomes more random without dimension reduction, implying that this stage is crucial for accurate instability ranking.

Scalability Analysis of CirSTAG Fig. 5 shows the runtime of CirSTAG across the nine benchmarks of varying complexity. CirSTAG demonstrates near-linear runtime, efficiently handling large circuit designs.

B. Stability Analysis under Circuit Topology Perturbations

In this case study, we investigate topology perturbations to demonstrate that **CirSTAG** effectively captures topology-related stability in functional reverse engineering tasks. We utilize datasets and a GNN model from [4], which employs a Graph Attention Network (GAT) for sub-circuit identification and labeling, achieving an accuracy of 98.87% on the interconnected dataset. In this model, graphs are derived from netlists, with nodes representing gates and edges depicting gate connections. Node features include surrounding gate information, detailing Boolean functionalities from gate inputs in the local neighborhood. Direct topology perturbations could mismatch

TABLE I: Comparison of average and maximum relative change (%) in arrival time prediction at primary output pin. Results for unstable and stable nodes are presented separately, denoted as 'unstable/stable'.

benchemark	Circuit stability analysis by CirSTAG with a GNN-based pre-routing timing analysis tool [17]											
	scale factor = 5×						scale factor = 10×					
	node perturb 5%		node perturb 10%		node perturb 15%		node perturb 5%		node perturb 10%		node perturb 15%	
	mean	max	mean	max	mean	max	mean	max	mean	max	mean	max
BM64	0.0915/ 0.0006	0.6328/ 0.0414	0.1391/ 0.0005	0.9016/ 0.0414	0.1563/ 0.0005	0.9828/ 0.0414	0.2057/ 0.0013	1.3715/ 0.0934	0.3125/ 0.0012	1.9934/ 0.0934	0.3504/ 0.0012	2.1751/ 0.0934
wbqspiflash	0.1106/ 0.0000	0.4078/ 0.0012	0.1090/ -0.0021	0.4798/ 0.0578	0.1081/ -0.0009	0.4798/ 0.0699	0.2479/ 0.0000	0.8934/ 0.0027	0.2443/ -0.0048	1.0382/ 0.1299	0.2434/ -0.0021	1.0382/ 0.1574
cic_decimator	0.3273/ 0.0000	0.9271/ 0.0000	0.3304/ 0.0000	0.9271/ 0.0029	0.3305/ 0.0022	0.9266/ 0.1047	0.7355/ 0.0000	2.1081/ 0.0000	0.7505/ 0.0001	2.1081/ 0.0069	0.7506/ 0.0049	2.1080/ 0.2361
des	0.1886/ 0.0030	1.4600/ 0.0848	0.2300/ 0.0030	1.9617/ 0.0848	0.2318/ 0.0030	1.9617/ 0.0848	0.4620/ 0.0068	3.4748/ 0.1907	0.5211/ 0.0068	4.6989/ 0.1907	0.5250/ 0.0068	4.6989/ 0.1907
aes_cipher	0.0753/ 0.0002	0.4464/ 0.0392	0.1265/ 0.0002	0.5755/ 0.0392	0.1665/ 0.0002	0.6023/ 0.0392	0.1698/ 0.0005	1.0109/ 0.0884	0.2839/ 0.0005	1.1368/ 0.0884	0.3744/ 0.0005	1.1921/ 0.0884
picorv32a	0.1505/ 0.0023	0.4522/ 0.1000	0.1540/ 0.0028	0.4781/ 0.1000	0.1559/ 0.0033	0.4781/ 0.1489	0.3383/ 0.0051	1.0095/ 0.2197	0.3461/ 0.0062	1.0587/ 0.2266	0.3501/ 0.0074	1.0587/ 0.3304
zipdiv	0.2136/ 0.0000	0.5189/ 0.0000	0.2202/ 0.0003	0.5807/ 0.0572	0.2211/ 0.0003	0.5807/ 0.0572	0.4769/ 0.0000	1.1610/ 0.0000	0.4905/ 0.0007	1.3093/ 0.1279	0.4924/ 0.0006	1.3039/ 0.1279
usb	0.2069/ 0.0029	0.4066/ 0.0737	0.2182/ 0.0043	0.4782/ 0.0945	0.2254/ 0.0096	0.4782/ 0.2194	0.4696/ 0.0065	0.9178/ 0.1673	0.4949/ 0.0099	1.0773/ 0.2142	0.5107/ 0.0217	1.0773/ 0.4862
usb_cdc_core	0.0090/ 0.0006	0.0894/ 0.0321	0.0165/ 0.0014	0.1550/ 0.0471	0.0198/ 0.0013	0.2164/ 0.0460	0.0196/ 0.0013	0.2031/ 0.0724	0.0373/ 0.0031	0.3448/ 0.1057	0.0449/ 0.0029	0.4879/ 0.1032

TABLE II: Comparison of average F1 macro score for sub-circuit classification under different level of perturbation ratio.

benchmarks	# nodes	# edges	F1 macro score						
			0%	5%		10%		20%	
			Original	Unstable	Stable	Unstable	Stable	Unstable	Stable
add mul combine	806	1640	0.9961	0.9555	0.9961	0.9184	0.9961	0.7002	0.9961
add mul comp	926	1829	1.000	0.9702	0.9961	0.9050	0.9963	0.5960	0.9961
add_mul_comp_sub	1161	2248	0.9864	0.9521	0.9809	0.9261	0.9774	0.8557	0.9673
add_mul_mix	1637	3299	0.6135	0.6044	0.6135	0.5789	0.6135	0.5323	0.6135
add_mul_sub	1111	2170	0.9748	0.9486	0.9712	0.9421	0.9691	0.7663	0.9656
add_mul	863	1709	0.9320	0.9001	0.9320	0.8690	0.9320	0.7870	0.9320

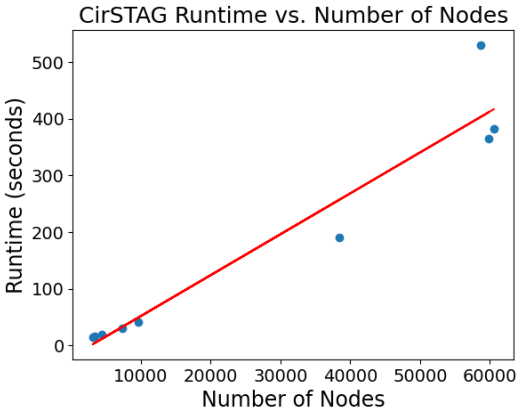


Fig. 5: Runtime scalability of CirSTAG

node features incorporating topological data. To address this, we perturb the topology while preserving feature relevance by clustering selected unstable or stable nodes based on input degrees and randomly swapping their output connections. This ensures that input topology and features remain valid post-perturbation. Analyzing the impact on sub-circuit classification using the F1 macro score, shown in Table II, we observe that as perturbation ratios increase, F1 macro scores decline more sharply for unstable nodes than for stable nodes. This suggests that unstable nodes, relying more on neighboring information, are more sensitive to topology changes, while stable nodes possess unique features supporting robust classification.

VI. Conclusion

We presented **CirSTAG**, a novel framework for quantifying the stability of circuit networks using GNNs. CirSTAG assesses circuit stability by estimating the impact of potential input perturbations on the GNN's output. By leveraging both input and output graph-based manifolds, our efficient spectral framework quantifies DMDs and assigns stability scores to nodes and edges, equivalent to the local Lipschitz constants of the GNN model. Empirical evaluations on pre-routing timing prediction and functional reverse engineering tasks demonstrate that CirSTAG effectively and scalably estimates circuit stability under various node feature and graph topology perturbations.

VII. Acknowledgments

This work is supported in part by the National Science Foundation under Grants CCF-2417619, CCF-2212370, and CCF-2205572.

REFERENCES

[1] Modeling and analysis of manufacturing variations. In *Proceedings of the IEEE 2001 Custom Integrated Circuits Conference (Cat. No. 01CH37169)*, pages 223–228. IEEE, 2001.

[2] A. Aghdaei and Z. Feng. Hyperef: Spectral hypergraph coarsening by effective-resistance clustering. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[3] L. Alrahis, J. Knechtel, and O. Sinanoglu. Graph neural networks: A powerful and versatile tool for advancing design, reliability, and security of ics. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pages 83–90, 2023.

[4] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu. Gnn-re: Graph neural networks for reverse engineering of gate-level netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2435–2448, 2022.

[5] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[6] F. Chen, S. Roch, K. Rohe, and S. Yu. Estimating graph dimension with cross-validated eigenvalues. *arXiv preprint arXiv:2108.03336*, 2021.

[7] Z. Chen, J. Xu, C. Alippi, S. X. Ding, Y. Shardt, T. Peng, and C. Yang. Graph neural network-based fault diagnosis: a review. *arXiv preprint arXiv:2111.08185*, 2021.

[8] W. Cheng, C. Deng, Z. Zhao, Y. Cai, Z. Zhang, and Z. Feng. Spade: A spectral method for black-box adversarial robustness evaluation. In *International Conference on Machine Learning*, pages 1814–1824. PMLR, 2021.

[9] T. Chu, Y. Gao, R. Peng, S. Sachdeva, S. Sawlani, and J. Wang. Graph sparsification, spectral sketches, and faster resistance computation via short cycle decompositions. *SIAM Journal on Computing*, (0):FOCS18–85, 2020.

[10] C. Deng, X. Li, Z. Feng, and Z. Zhang. Garnet: Reduced-rank topology learning for robust and scalable graph neural networks. *arXiv preprint arXiv:2201.12741*, 2022.

[11] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst. Learning laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23):6160–6173, 2016.

[12] X. Dong, D. Thanou, M. Rabbat, and P. Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63, 2019.

[13] H. E. Egilmez, E. Pavez, and A. Ortega. Graph learning from data under laplacian and structural constraints. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):825–841, 2017.

[14] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.

[15] Z. Feng. Sgl: Spectral graph learning from measurements. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 727–732. IEEE, 2021.

[16] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[17] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1207–1212, 2022.

[18] V. Kalofolias. How to learn a graph from smooth signals. In *Artificial Intelligence and Statistics*, pages 920–929, 2016.

[19] B. Lake and J. Tenenbaum. Discovering structure by learning sparse graphs. 2010.

[20] Y. P. Liu, S. Sachdeva, and Z. Yu. Short cycles via low-diameter decompositions. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2602–2615. SIAM, 2019.

[21] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 733–738. IEEE, 2021.

[22] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[23] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

[24] H. Ren, S. Nath, Y. Zhang, H. Chen, and M. Liu. Why are graph neural networks effective for eda problems? In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–8, 2022.

[25] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[26] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[27] R. R. Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018.

[28] N. Wu, Y. Li, C. Hao, S. Dai, C. Yu, and Y. Xie. Gamora: Graph learning based symbolic reasoning for large-scale boolean networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[29] B. Yu, H. Yin, and Z. Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[30] Y. Zhang, Z. Zhao, and Z. Feng. Sf-sgl: Solver-free spectral graph learning from linear measurements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[31] Z. Zhao, Y. Zhang, and Z. Feng. Towards scalable spectral embedding and data visualization via spectral coarsening. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 869–877, 2021.

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

Grasp: Group-based Prediction of Activation Sparsity for Fast LLM Inference

Jiho Shin
University of Seoul
Seoul, South Korea
sjh010529@uos.ac.kr

Hoeseok Yang*
Santa Clara University
Santa Clara, CA, USA
hoeseok.yang@scu.edu

Youngmin Yi*
Sogang University
Seoul, South Korea
ymyi@sogang.ac.kr

***Abstract*—Optimizing LLM inference has become increasingly important as the demand for efficient on-device deployments grows. To reduce the computational overhead in the MLP components, which account for a significant portion of LLM inference, ReLU-fied LLMs have been introduced to maximize activation sparsity. Several sparsity prediction methods have been developed to efficiently skip unnecessary memory accesses and computations by predicting activation sparsity. In this paper, we propose a novel magnitude-based, training-free sparsity prediction technique called Grasp that builds on the existing sign bit-based method for ReLU-fied LLMs. The proposed method enhances prediction accuracy by grouping values considering the distribution within vectors and explicitly accounting for statistical outliers. This allows us to estimate the impact of each element more accurately yet in an efficient way, improving both activation sparsity prediction accuracy and computational efficiency. Compared to the-state-of-the-art technique, Grasp achieves higher sparsity prediction accuracy and 11% higher skipping efficiency, which corresponds to 1.85× speedup against the dense inference.**

***Index Terms*—Large language models, fast inference, activation sparsity, sparsity prediction**

I. Introduction

Following the success of Transformer-based models like GPT [1], there has been a surge in demand for performing the inference of Large Language Models (LLMs) directly on devices rather than offloading to the cloud. Empirical studies indicate that Multilayer Perceptron (MLP) layers dominate the computational load in Transformers, accounting for approximately 60-70% of the total FLOPs in models like GPT-3 [2] and also 60% of the total latency in models like Llama2 [3] with the default sequence length of 4K. As a result, recent studies have focused on maximizing activation sparsity within the MLP layers to improve efficiency in both memory access and computation.

Notably, since the popularly used activation function in MLP blocks, such as SiLU (Sigmoid Linear Unit) [4] or GELU (Gaussian Error Linear Unit) [5], do not produce a significant number of zero activations, a new approach, called *ReLU-fication* [6], has emerged. This involves replacing the activation function in pre-trained LLMs with sparsity-inducing functions like ReLU, followed by fine-tuning. This method has been shown to enhance memory and computational efficiency while maintaining nearly the same level of accuracy. Building on this foundation, a number of studies [7]–[10] have proposed optimizing MLPs by predicting activation sparsity in advance.

These techniques are based on predicting which parts of the MLP weight matrix can be skipped depending on the values of the hidden state. For instance, `DejaVu` [7] achieves this by adding two low-rank fully connected layers to the MLP block to predict a sparsity vector. Thus, it comes with the drawback of incurring large overhead during inference due to the need to run the predictor. More critically, it requires retraining the predictor whenever the model is modified.

To overcome this, a training-free activation sparsity prediction technique, `SparseInfer` [11], has recently been proposed [11]. In this approach, the prediction leverages the property that the values of hidden states and weights in LLMs typically follow a normal distribution. By simply looking up the sign bits of these values, it efficiently determines which elements will be zeroed out in the ReLU-fied LLM, thus predicting activation sparsity with significantly lower overhead and without the need for additional training.

However, this technique has limitations in prediction accuracy since it relies on inspecting only a small portion of the information, i.e., sign bits. As a result, it is required to adjust the conservativeness of predictions for some early layers to compensate for inaccuracies. In this work, we propose an improved method that enhances prediction accuracy while still ensuring high predictor efficiency and achieving a high degree of activation sparsity.

In this work, we propose a group-based activation sparsity prediction (`Grasp`), which improves the accuracy of sparsity prediction by considering the distribution of hidden states and weight values in the MLP layers. Unlike `SparseInfer`, which predicts the result of a vector's inner product by simply determining whether each partial multiplication result is positive or negative and then counting the occurrences of positive and negative values, `Grasp` quantifies the contribution of each partial product differently by grouping them appropriately. In addition, to further enhance prediction accuracy, we extend our prediction to account for statistical outliers.

The key to improving prediction accuracy in the proposed method lies in *grouping* the weights or hidden state values based on their distribution and quantifying the impact of each group differently. We demonstrate that even a simple logistic regression model calibrated on a small data can achieve substantial improvements in the activation sparsity prediction accuracy. Furthermore, we show that the precision and recall of the prediction are closely correlated with the quality and speed of LLM inference, respectively. By effectively leveraging this trade-off, we derive an activation sparsity predictor that is both accurate and efficient, optimizing inference performance without sacrificing model quality.

The proposed activation sparsity predictor was implemented and verified for ProSparse-Llama-13B, a ReLUfied version of Llama-2 13B, on a mobile grade GPU, Jetson Orin AGX 64GB [12], using an open-source LLM inference engine, llama.cpp [13]. As a result, we achieved a 1.85× speedup compared to the dense version without any loss in accuracy. Notably, this performance surpasses the state-of-the-art [11] by 11% in inference speed. Additionally, our method exhibited approximately 2× lower overhead than the training-based activation sparsity predictor [7], further demonstrating its efficiency.

*Youngmin Yi and Hoeseok Yang are corresponding authors.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (No. 2022R1F1A1075190)

II. Background: Activation Sparsity Prediction

In this section, we briefly review the activation sparsity prediction and utilization of `SparseInfer` [11]. While we present it with a gate-based MLP which is popularly used in llama-baed LLMs for simplicity, it can be generally applicable to any ReLU-fied LLMs.

Activation Sparsity in MLP Operations: The hidden state $X \in \mathbb{R}^{1\times d}$ undergoes two independent linear transformations using $W_{gate} \in \mathbb{R}^{d\times k}$ and $W_{up} \in \mathbb{R}^{d\times k}$, respectively, to obtain two hidden states as follows:

$$h_1 = \sigma(X \cdot W_{gate}) \tag{1}$$

and

$$h_2 = X \cdot W_{up}, \tag{2}$$

where $\sigma(\cdot)$ denotes a ReLU activation function. Then, another hidden state is obtained by element-wise multiplication between h_1 and h_2, which is subsequently linearly transformed by $W^T_{down} \in \mathbb{R}^{k\times d}$ to produce the final output of the MLP[1]:

$$MLP(X) = (h_1 \odot h_2) \cdot W^T_{down}, \tag{3}$$

where and $\odot$ represents element-wise multiplication.

Note that any negative value in $X \cdot W_{gate}$ will result in 0 in h_1 due to the ReLU activation σ. Consequently, the corresponding elements in $h2 = X \cdot W_{up}$ will never be utilized in the subsequent computations and can thus be skipped. Similarly, the associated rows in W^T_{down} can be safely ignored without affecting the final output. In other words, if we can effectively estimate which elements of h_1 will be zero, the linear transformation overheads in equations (2) and (3) can be significantly reduced. This optimization technique is referred to as activation sparsity.

Sign-bit Based Sparsity Prediction: For the ith row (or column) of W_{gate} (or W^T_{down}) to be skipped due to activation sparsity, the inner product between X and W^i_{gate} (the ith row of W_{gate}) must be negative. In [11], this is predicted by examining the most significant bits (sign bits) of the corresponding elements in X and W^i_{gate}. At each position, the two sign bits are XORed; a result of 1 indicates that the signs are different, meaning the product would be negative. The method counts the number of such occurrences in N_{neg} and, if this count exceeds the number of positions where the product would be positive ($N_{pos} = d - N_{neg}$), it predicts the inner product as negative.

This activation sparsity prediction can be inherently inaccurate because it does not consider the absolute values; it only assesses whether the element-wise multiplication results in positive or negative contributions to the final inner product result. Due to this limitation, inaccuracies are inevitable, potentially impacting the quality of LLM inference. To mitigate this effect, the activation sparsity condition is adjusted using a tunable parameter α as follows:

$$N_{neg} > \alpha \cdot N_{pos} = \alpha \cdot (d - N_{neg}), \tag{4}$$

where $\alpha > 1.0$ results in a more conservative activation sparsity prediction, thereby reducing the degree of sparsity.

Note that there is still an opportunity to leverage sparsity even if the sparsity prediction for $X \cdot W_{gate}$ is incorrect. For instance, let us suppose that $X \cdot W^i_{gate}$ was incorrectly predicted as positive, but upon actual computation, $X \cdot W^i_{gate}$ turns out to be negative, resulting in $\sigma(X \cdot W^i_{gate}) = 0$. Then, it is possible to skip further computations in the following linear transformations and element-wise multiplication, for the ith row (or column) of X_{up} (or X^T_{down}), thus still achieving some degree of sparsity optimization post-prediction. This is referred to as *actual*, as opposed to *predicted*, sparsity in [11].

[1] d and k are the hidden state and intermediate dimensions. In Llama-2, they are $5,120$ and $13,824$, respectively.

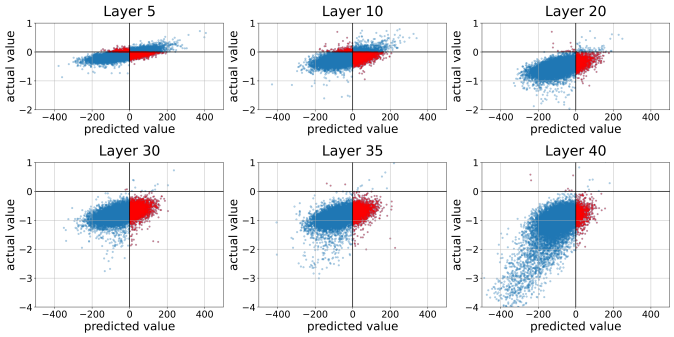


Fig. 1. Sparsity prediction results of `SparseInfer` [11]: predicted value $N_{pos} - N_{neg}$ (X-axis) and the actual $X \cdot W_{gate}$ value (Y-axis).

Precision and Recall: It is noteworthy that the accuracy of activation sparsity prediction not only impacts the quality of LLM inference but also affects inference speed. The precision of the activation sparsity predictor indicates the proportion of cases predicted as sparse that can actually be skipped. Low precision degrades the quality of LLM inference, as incorrect skips introduce errors that cannot be corrected later. In contrast, the recall of the activation sparsity predictor measures how effectively the predictor identifies all feasible cases for skipping. A Low recall indicates that a significant portion of genuinely skippable computations were not identified by the predictor, thereby slowing down inference. However, this does not degrade the quality of LLM inference. Additionally, cases that were falsely predicted as non-sparse can still benefit from exploiting the *actual* sparsity during subsequent computations.

III. Analysis of Sparsity Prediction Accuracy

To enhance the accuracy of `SparseInfer`, the sign-bit-based activation sparsity predictor proposed in [11], we conducted an analysis of its sparsity prediction performance. Fig. 1 illustrate the scatter plots depicting the layer-wise sparsity prediction results for the ProSparse-13B model [9] using the GSM8K dataset [14]. In these 2-D charts, each point represents a sparsity prediction result, where the X-axis indicates the predicted value, $N_{pos} - N_{neg} = d - 2N_{neg}$, while the Y-axis shows the actual inner product results, $h_1 = X \cdot W_{gate}$. On the X-axis, higher (positive) values correspond to a lower probability of being sparse, while smaller (negative) values indicate a higher likelihood of sparsity. On the other hand, the Y-axis represents the actual outcome of activation sparsity, where negative values imply activation sparsity. Therefore, we can evaluate the accuracy of the sparsity prediction using the four quadrants of the scatter plot as follows:

- (top-right) first quadrant: predicted as non-sparse and actually non-spares (true negative),
- (top-left) second quadrant: predicted as sparse but actually non-sparse (**false positive**),
- (bottom-left) third quadrant: predicted as sparse and actually sparse (true positive), and
- (bottom-right) fourth quadrant: predicted as non-sparse but actually sparse (**false negative**).

The inaccuracies in the sparsity prediction of [11] can be categorized into two types, both of which are highlighted as red points in Fig. 1 for clarity. False positives are represented by points plotted in the second quadrant, which negatively affect precision. In contrast,

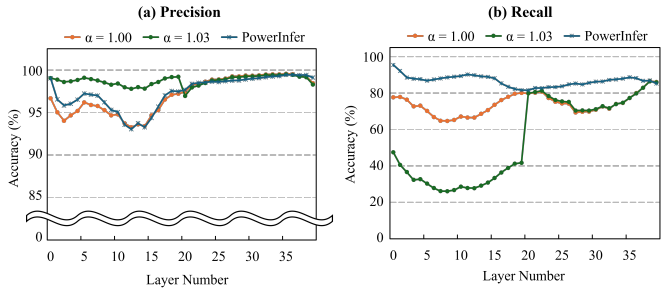


Fig. 2. Activation sparsity prediction results of `PowerInfer` [8] and `SparseInfer` [11] ($\alpha = 1.0$ and $\alpha = 1.03$) for 10 GSM8K dataset samples.

false negatives appear in the fourth quadrant, impacting recall. However, lower recall does not degrade the quality of LLM inference; rather, it only affects speed due to unnecessary computations. This impact on speed can be partially alleviated through the use of *actual* sparsity, as previously discussed.

However, as formulated in equation (4), the activation sparsity prediction is based on $\alpha \cdot N_{pos} - N_{neg}$ rather than $N_{pos} - N_{neg}$. Therefore, using a value of $\alpha > 1.0$ for the early layers, as proposed in [11], has the effect of shifting the points plotted in the scatter plots of Fig. 1 horizontally to the right. By doing so, the false positives in the second quadrant are shifted into the first quadrant, where they are correctly predicted as true negatives, thus improving precision. However, this horizontal shift introduces a side effect: a significant portion of the true positives in the third quadrant are moved into the fourth quadrant, turning them into false negatives. As a result, this leads to a reduction in recall.

This trade-off between precision and recall becomes more apparent when comparing various activation sparsity techniques. Fig. 2 illustrates the differences in precision and recall between two approaches: the `PowerInfer` method [8], which uses an FFN-based predictor, and two versions ($\alpha = 1.0$ and $\alpha = 1.03$) of `SparseInfer` [11]. `SparseInfer` [11] with the default setting ($\alpha = 1.0$) had significantly lower overhead but performed worse than `PowerInfer` [8] in both precision and recall. When $\alpha = 1.03$ is applied to the early layers, the precision has successfully increased precision beyond that of `PowerInfer`. However, as explained earlier, this came at the cost of reduced recall, demonstrating the inherent trade-off between precision and recall. In this paper, we aim to overcome the limitation by improving precision without significantly sacrificing recall.

The scatter plots in Fig. 1 indicate that sign-based predictions often result in false negatives (bottom-right, the fourth quadrant), particularly in deeper layers, where actual values are predominantly negative. Also, we observe that $X \cdot W_{gate}$ values tend to grow in magnitude as the layer gets deeper. By factoring in the absolute value of $X \cdot W_{gates}$ products alongside their signs, the prediction errors can be mitigated, improving both efficiency and accuracy.

IV. Grasp: Group-based Activation Sparsity Prediction

To develop an effective predictor, we need to (1) leverage the simplicity of the baseline sign-based predictor [11], (2) use the absolute values of $X \cdot W$ to refine predictions, and (3) account for activation outliers, particularly in problematic layers, to balance precision and recall effectively. This approach aims to combine the strengths of the sign-based [11] and training-based [8] predictors, achieving the desired trade-off between accuracy and inference efficiency. This improved prediction is illustrated in the left part of Fig. 3, while the utilization of sparsity is depicted in the right part. This example includes all four prediction cases (false/true negatives and positives) explained in Section III. As demonstrated in the experiments, the proposed method can effectively reduce false negatives/positives.

A. Magnitude Consideration

As discussed in the previous section, `SparseInfer` only utilized the sign bits of X and W, disregarding the magnitude of the values. This limitation hindered its ability to improve both precision and recall. In this work, we incorporate magnitude information in addition to the sign bits. To maintain efficient prediction, we group the products and assign a representative magnitude to each group. This approach enhances both accuracy and speedup while still avoiding direct computation of the inner product.

To group each product by magnitude, it is first necessary to categorize the magnitudes of the operands, X and W, which make up the product. Unlike sign bits, magnitude information cannot be directly extracted from X and W, thus requiring the establishment of a criterion for the grouping. As observed in previous studies [11], [15], the input distribution is similar to a normal distribution. In this work, assuming the input distribution follows a normal distribution, we classify magnitudes using quartiles. Specifically, since X and W are assumed to have an actual mean close to 0 and follow a normal distribution, elements are classified as follows: if they fall within Q_2 and Q_3 quartiles, meaning their absolute value is smaller than Q_3 point (the upper 25%), they are assigned to the *small* group. If they fall within Q_1 and Q_4 quartiles, meaning their absolute value is larger than Q_3 point, they are assigned to the *large* group.

Sorting vectors to determine classification points would incur significant overhead. Instead, since X and W exhibit distributions close to normal, the quartile boundaries for any x_i or w_i can be efficiently estimated using the mean and standard deviation as shown in the following formula[2].

$$Q_3 = \mu + 0.6745 \cdot \sigma \ , \quad Q_1 = \mu - 0.6745 \cdot \sigma \tag{5}$$

Once the groups for x_i and w_i are determined, the product group is defined as follows: If both x_i and w_i belong to the large group, the product is assigned to Group G_1. If one of them belongs to the large group and the other to the small group, the product is assigned to Group G_2. If both belong to the small group, the product is assigned to Group G_3.

After assigning each product to a group, the sign bits of the products within each group are examined to count which sign ($+$ or $-$) occurs more frequently. This count is then multiplied by the representative magnitude of the respective product group, yielding a weighted sum. If this weighted sum is 0 or negative, the corresponding element of the h_1 is predicted to be sparse. In essence, while this approach avoids directly computing the inner product using the actual x_i and w_i, it approximates the process by incorporating magnitude information via representative values for each group. Note that the number of groups used introduces a trade-off between prediction overhead and precision/recall performance.

Algorithm 1 outlines the prediction process used by `Grasp`. Here, N_i represents the difference between the positive and negative counts ($N_{\text{pos}} - N_{\text{neg}}$) for each group. To update N_i, the signs of all X and W values in the group are XORed. Finally, the differences in sign counts for each group are weighted by S_i, the representative scale for the group, to predict the Activation Sparsity. Determining each group's scale S_i will be explained in Sec. IV-C.

[2]In a normal distribution, Q_3i s calculated as $\mu + 0.6745 \cdot \sigma$, where 0.6745 is the Z-score corresponding to the cumulative probability of 0.75.

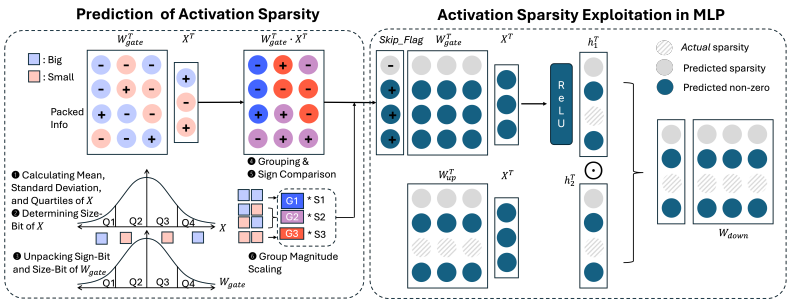


Fig. 3. Overview of the proposed group-based activation sparsity prediction.

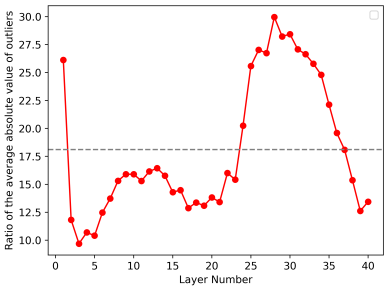


Fig. 4. Ratio of the average absolute value of outliers to that of non-outliers in GSM8K.

Algorithm 1 Activation Sparsity Prediction Based on Groups

Input: Vectors $x \in \mathbb{R}^d$ and $W_{gate,i} \in \mathbb{R}^d$
Output: Sparsity Detected ($N_{total} \leq 0$) or Not ($N_{total} > 0$)
for $i = 1$ to 3 **do**
  $G_i \leftarrow [0, 0, \ldots, 0]$ ▷ array for group index
  $N_i \leftarrow [0, 0, \ldots, 0]$ ▷ array for $N_{pos} - N_{neg}$
end for
for $i = 1$ to d **do**
  if big($x[i]$) **and** big($W[i]$) **then**
    $G_1[i] \leftarrow 1$ ▷ big(x) returns true if $x < Q_1$ or $Q_3 < x$
  else if small($x[i]$) **and** small($W[i]$) **then**
    $G_3[i] \leftarrow 1$ ▷ small(x) = !big(x)
  else
    $G_2[i] \leftarrow 1$
  end if
end for
for $i = 1$ to d **do**
  Extract the sign bit of $x[i]$ as $s_x \leftarrow \text{MSB}(x[i])$
  Extract the sign bit of $W[i]$ as $s_W \leftarrow \text{MSB}(W[i])$
  $s \leftarrow s_x \oplus s_W$ ▷ $s = 0$ for the same signs, otherwise $s = 1$
  for $j = 1$ to 3 **do**
    if $G_j[i] = 1$ and $s = 1$ **then**
      $N_j \leftarrow N_j - 1$ ▷ product is a negative number
    else if $G_j[i] = 1$ and $s = 0$ **then**
      $N_j \leftarrow N_j + 1$ ▷ product is a positive number
    end if
  end for
end for
$N_{total} = \sum_{i=1}^{3} S_i \cdot N_i$

B. Outlier Consideration

The group assignment and the representative magnitude values allow significantly more accurate prediction than the sign-bit-based prediction while still maintaining high efficiency. However, in certain models, we observed that a relatively large number of outliers with very high absolute values made accurate prediction challenging. Without explicitly accounting for these outliers, the precision could drop significantly.

In this work, outliers are defined as values with a z-score exceeding 6. Assuming the input values follow a perfect normal distribution, with variance σ^2 (estimated between 0.02 and 0.4 from GSM8K data) and dimensionality of 5k (e.g., in Llama2), such extreme values are so rare that they did not appear in 10,000 simulation runs. However, in practice, for X, we found an average of 5 elements per layer with a Z-score exceeding 6. When these outliers are simply categorized into the large product group (i.e., G_1), prediction accuracy will deteriorate.

Although there exist only about 5 elements on average in a 5k-dimensional X vector in each layer, their impact is significant. As shown in Fig. 4, the ratio of the average absolute value of outliers to that of non-outliers varies across layers but is approximately 18.5× when the GSM8K dataset is used. Moreover, when comparing the recall graph of `SparseInfer` with the ratio of outlier to non-outlier magnitudes, we observed that layers with relatively lower recall (e.g., around layer 10 and around layer 30) tended to exhibit larger magnitude differences between outliers and non-outliers. This observation highlights the significant influence of outliers on prediction performance.

How, then, should these extreme outliers with large absolute values be handled? One possible solution could involve directly calculating the actual product by multiplying the corresponding x_i with w_i. However, note that during the prediction phase, the W matrix is not loaded into memory. Instead, only a compact matrices containing pre-packed sign bits and magnitude bits for w_i are available, which will be explained in detail in Sec. IV-D. Given the high cost of loading the entire W matrix, computing the actual product is not an efficient choice. To address this, we introduce a new product group specifically for outliers and assign a representative scale to this group. The outlier group is defined as $G_0 = \{x \,|\, |x - \mu| > 6\sigma\}$, which includes values in X with a z-score greater than 6.

Previously, we discussed how the number of groups creates a trade-off between prediction accuracy and computational efficiency. Although it would be possible to divide outliers into multiple groups based on different ranges, this work designates all outliers as a single group. This approach balances prediction complexity and efficiency while explicitly accounting for the significant influence of outliers.

C. Determining Impact of Each Factor

Determining the representative magnitude for each group can be approached in various ways, and the choice of these values significantly impacts prediction accuracy. This challenge becomes especially pronounced for outliers, where the group contains relatively few elements with high variability, making it more difficult to define an appropriate representative value. In this work, we propose the following two methods.

1) The Average Absolute Value Ratio : The representative value for each group should be relative to those of other groups, reflecting the magnitude characteristics of its elements. To achieve this, we calculate the average absolute value of the elements ($|x_i \cdot w_i|$) belonging to each group.

The scale for the group with the smallest average absolute value, G_3, is set to $S_3 = 1$. For all other groups, the scale S_i is determined as the ratio of the average absolute value of elements in group G_i to the average absolute value of elements in G_3:

$$S_i = \frac{\text{Mean}\left(|x_i \cdot w_i| \in G_i\right)}{\text{Mean}\left(|x_i \cdot w_i| \in G_3\right)}. \tag{6}$$

2) Logistic Regression: Since this is a binary classification problem (sparse vs. non-sparse) based on a linear combination of independent variables, Logistic Regression (LR) is well-suited for the task. The independent variables are N_i, and to improve the accuracy of the LR model, the N_i values are standardized using a calibration dataset. Standardization reduces differences in scale among the variables, allowing the model to perform better. In this setup, the group-specific weights S_i serve as the regression coefficients.

LR essentially computes a linear combination of N_i and S_i, which is passed through a sigmoid function to produce a value between 0 and 1, where Class 0 represents non-sparse elements and Class 1 represents sparse elements. The classification threshold th determines the decision boundary: a higher th leads to more elements being classified as Class 0 (non-sparse), favoring conservative predictions, while a lower th results in more elements classified as Class 1 (sparse), favoring aggressive predictions.

This trade-off between precision and recall allows tuning based on the desired balance. By default, th is chosen to achieve the most balanced trade-off between recall and precision. Using the ROC curve of one of the calibration datasets, th is set to the point where Youden's J statistic [16] is maximized.

D. Prediction Overhead

1) Packing of W_{gate} sign/magnitude information (Offline): W_{gate}, the model's weights, remain fixed once loaded into memory. As shown in Fig. 3, during model loading or as a pre-processing, `Grasp` extracts and pre-computes the sign and magnitude information of W_{gate} and packs them into separate matrices, where the sign and magnitude information is represented in one bit, respectively. The sign bits are packed into 32-bit integers. For the magnitude bit assignment, the mean and standard deviation are first calculated, and then the elements are divided into quartiles. Elements with absolute values larger than Q_3 are assigned one, otherwise zero. These one-bit values are also packed into 32-bit integers.

2) Calculating mean and variance of X (Online): Grasp's predictor extracts both the sign bits and magnitude bits of X during runtime. To obtain the magnitude information, it calculates the mean and standard deviation of X. Given that X has a relatively small dimension (e.g., 5k in Llama2-13B), GPUs can efficiently handle this computation in parallel, minimizing the overhead. For example, this magnitude extraction accounts for approximately 11% of the total overhead of Grasp's predictor, which is about 2.4% of MLP computation time on Jetson Orin AGX.

3) Efficient grouping using bitwise operations: The GPU kernel is provided with the two-bit information for W computed offline, and the three-bit information for X computed online: the sign bit, magnitude bit, and outlier bit indicating whether each element of X is classified as an outlier.

The groups G_1, G_2, and G_3 are defined based on the magnitude bits of X and W, and can be efficiently determined using bitwise operations: G_1 is determined using bitwise AND operation; G_2 using bitwise XOR operations; G_3 using the complement of G_1 AND G_2. These bitwise operations allow efficient classification of the products into their respective groups, enabling accurate and efficient predictions during runtime.

4) Calibration for scale values: Both the scaling methods, Average Value Ratio and Logistic Regression (LR), entail very minimal overhead. Although this work calibrates these scales offline, it is entirely feasible to perform calibration online. For online calibration, `Grasp` performs inference in the dense mode for a short duration, equivalent to processing tens to around 100 tokens. For LR, `Grasp` simply stores the results of the existing inner products in a separate memory space. Then, it determines the scaling factors S_i by LR based on these results. This process is computationally light and can be completed in a matter of seconds. The Average Value Ratio method, unlike LR, does not require a training-like process. It only calculates the average absolute value of the product terms in each group. While this requires an additional calculation step to compute group-wise absolute value means, it avoids the fitting process required in LR. Note that, once the scaling factors S_i are determined, no further calibration is required for for the remainder of the workload.

V. Experiments

A. Experimental Setup

1) Models and Datasets: We used ProSparse-13B and ProSparse-7B models [9], which are based on Llama2 with ReLU activation functions. To ensure a fair comparison with [11], we used the same benchmark, GSM8K [14] and BBH [17] test datasets, to measure the accuracy. The inference speed was measured using randomly extracted 20 samples from GSM8K test dataset (with seed 42). For calibrating the scale values of Grasp, randomly selected 128 samples from the GSM8K train dataset were used. This calibration set was used for two different methods discussed in Sec. IV: the Average Absolute Value Ratio (denoted as *R*) and Logistic Regression (denoted as *LR*). Among the extracted calibration set, one sample is reserved as a validation set to determine the regression threshold th in LR.

2) Frameworks and Hardware: As in previous studies [8], [11], `Grasp` is implemented based on the open-source inference engine Llama.cpp [13]. The measurements of inference speed were conducted on a mobile-grade GPU, Jetson Orin 64GB, on Ubuntu 22.04 with JetPack 6.0 DP and CUDA 12.2. We used an LLM benchmark suite, llm-harness [18], to evalute the inference accuracy. Since it does not directly support Llama.cpp, accuracy measurements were separately performed using a modified PyTorch-based Transformers framework implemented with the same principles. For the same reason, we could not evaluate the accuracy of `PowerInfer` using llm-harness. Instead, we conducted a evaluation focusing solely on the precision and recall metrics, as shown in Fig. 2.

B. Impact of Group-based Prediction

As a baseline for comparison, we use the default Llama.cpp implementation without considering sparsity. Additionally, for a comparative evaluation, we compare the proposed method against the previous research, `PowerInfer` [8] and two versions of `SparseInfer` [11]: $\alpha = 1.0$ and $\alpha = 1.03$. For the proposed method, we conducted an ablation study by separating two variations: one that implements only Magnitude Consideration (*MC*) and another that additionally incorporates Outlier Consideration (*MC + OC*). Furthermore, as stated earlier, we evaluated both approaches using two different methods for determining scale values, *R* and *LR*.

1) Accuracy: Let us first directly compare the performance of sparsity prediction. Fig. 5 is a scatter plot comparing the predictions and actual values of the `Grasp MC + OC R` prediction method, obtained using the same approach as Fig. 1. Compared to `SparseInfer`, a noticeable reduction in false negatives in the fourth quadrant can be observed, which is the primary reason for the performance improvement of `Grasp`.

Tables I and II present the scores obtained for each approach on GSM8K and BBH. `Grasp MC + OC` demonstrated overall better performance compared to `Grasp MC`. Regardless of the method for calibrating the scale factors, the accuracy drop for both models

TABLE I
ACCURACY COMPARISON OF PROSPARSE-LLAMA2-13B

Method	Scale	GSM8K	BBH	Average
Baseline	-	31.92	44.88	38.40
[11] (α = 1.00)	-	29.57 (-2.35)	42.87 (-2.01)	36.22 (-2.18)
[11] (α = 1.03)	-	31.39 (-0.53)	44.71 (-0.17)	38.05 (-0.35)
Grasp MC	R	31.16 (-0.76)	44.02 (-0.86)	37.59 (-0.81)
	LR	30.33 (-1.59)	43.96 (-0.92)	37.15 (-1.25)
Grasp MC + OC	R	30.93 (-0.99)	44.86 (-0.02)	37.90 (-0.50)
	LR	32.15 (+0.23)	44.45 (-0.43)	38.30 (-0.10)

TABLE II
ACCURACY COMPARISON OF PROSPARSE-LLAMA2-7B

Method	Scale	GSM8K	BBH	Average
Baseline	-	13.57	35.94	24.76
[11] (α = 1.00)	-	8.95 (-4.62)	27.60 (-8.34)	18.28 (-6.48)
[11] (α = 1.03)	-	12.59 (-0.98)	35.68 (-0.26)	24.14 (-0.62)
Grasp MC	R	10.99 (-2.58)	33.67 (-2.27)	22.33 (-2.43)
	LR	11.98 (-1.59)	28.14 (-7.80)	20.06 (-4.70)
Grasp MC + OC	R	13.34 (-0.23)	35.29 (-0.65)	24.32 (-0.44)
	LR	13.19 (-0.38)	35.08 (-0.86)	24.14 (-0.62)

remained within a margin of less than 1%p. Also, it has been observed that `Grasp` achieved higher average accuracy than the best accuracy obtained by `SparseInfer` ($\alpha = 1.03$). It is noteworthy that the accuracy drop on BBH was maintained within a margin of less than 1%p even if the calibration data was based on GSM8K.

2) End-to-End Speedup: Fig. 6 shows the average end-to-end decoding speedup for generating a single token with each approach. `SparseInfer` had limitations in effectively balancing the trade-off between accuracy and inference speed. Specifically, while $\alpha = 1.0$ was the fastest, it resulted in a significant drop in accuracy. On the other hand, with $\alpha = 1.03$, accuracy improved, but it became the slowest among the proposed methods, except for `PowerInfer`. In contrast, `Grasp` explored this trade-off more precisely. Particularly, with `Grasp MC + OC`, the 13B model achieved up to 11% faster inference compared to `SparseInfer` with $\alpha = 1.03$, while maintaining nearly the same accuracy level.

In the ProSparse-13B model, the MLP block accounts for approximately 62% of the total computation. Assuming an average sparsity of 88% in the model [9], ideal computation in the MLP block without predictor overhead would yield a maximum performance improvement of 2.08×. This means that a maximum of 25% improvement in performance is achievable compared to `SparseInfer` $\alpha = 1.03$. Furthermore, `Grasp MC + OC (R)` achieved approximately 89% of the maximum performance improvement potential, reaching a 1.85× speedup compared to the baseline.

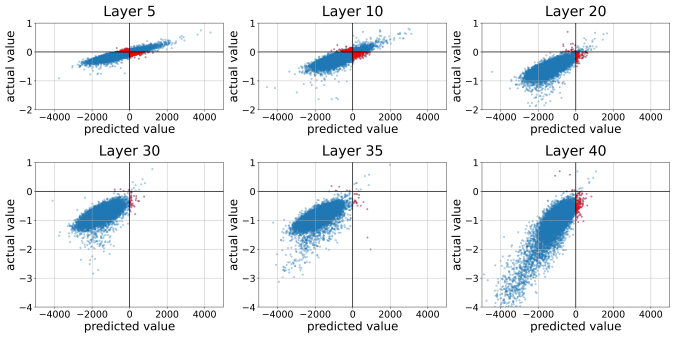


Fig. 5. Sparsity prediction results of `Grasp MC + OC (R)`: predicted value $\sum_{i=0}^{3} S_i \cdot N_i$ (X-axis) and the actual $X \cdot W_{gate}$ value (Y-axis).

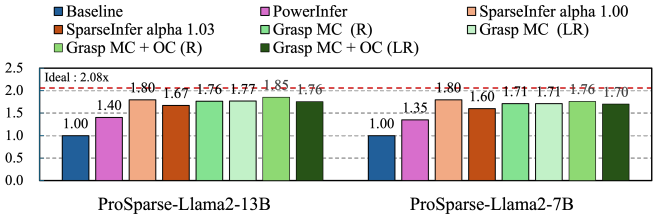


Fig. 6. End-to-end decoding speedup for each approach (baseline = 160.6ms)

TABLE III
ACCURACY AND LATENCY OF PROSPARSE MODELS WITH DIFFERENT THRESHOLDS IN LOGISTIC REGRESSION WITH GRASP

Model	th	GSM8K	BBH	Average	E2E latency
13B	**1.00x**	32.15	44.45	38.30	91.49ms
	0.95x	31.69 (-0.46)	44.16 (-0.29)	37.93 (-0.37)	88.57ms
	0.90x	30.48 (-1.67)	44.43 (-0.02)	37.46 (-0.84)	87.79ms
7B	**1.00x**	13.19	35.08	24.14	50.49ms
	0.95x	12.59 (-0.60)	34.11 (-0.97)	23.35 (-0.79)	46.75ms
	0.90x	11.75 (-1.44)	33.13 (-1.95)	22.44 (-1.70)	46.37ms

Note that the end-to-end latency shown in Fig. 6 includes the predictor overhead: 10ms for `PowerInfer`, 2.8ms for `SparseInfer`, and 5.5ms for `Grasp`, for the ProSparse-13B model.

C. Impact of tunable threshold

The threshold th in LR plays the role of a tunable parameter for accuracy-speed trade-off. After LR fitting, Youden's J statistic [16] yields optimal th for each layer. Lower th means larger ones (i.e., sparsity), indicating aggressive prediction. In order to show the effectiveness of this tunable threshold, we explored more aggressive sparsity predictor variations from `Grasp MC + OC`; we scaled down the threshold th to 0.95 and 0.90. Table III shows the accuracy and end-to-end decoding latency per token of `Grasp MC + OC` with these various threshold values. As th decreased, latency was improved, but accuracy also tended to decline. When the threshold is scaled down to 0.9×, it shows 1.04-fold speedup over the original threshold (1.82-fold speedup over the baseline). This suggests that th has potential as a control knob to adjust the trade-off between speed and accuracy. For the 13B model, accuracy was maintained within a margin of less than 1%p, while for the 7B model, the accuracy drop was bigger.

VI. CONCLUDING REMARKS

In this paper, we proposed a group-based enhanced activation sparsity prediction method of ReLU-fied LLMs. Unlike the existing sign-based prediction method that relies solely on the signs of operands, the proposed approach groups the inner product operands based on the relative magnitudes and assigns different weights to each group, thereby improving prediction accuracy. Additionally, statistical outliers were considered in the prediction to improve the prediction accuracy further. Experimental results demonstrated that the overhead introduced by the proposed predictor was minimal, and it effectively explored the trade-off between the accuracy and speed of LLM inference. Although the proposed method requires minimal sample-based scale factor calibration, it offers significant accuracy improvements with low overhead, making it promising for on-the-fly calibration in practical applications.

REFERENCES

[1] A. Radford, "Improving language understanding by generative pre-training," 2018.

[2] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[4] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[5] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[6] I. Mirzadeh, K. Alizadeh, S. Mehta, C. C. D. Mundo, O. Tuzel, G. Samei, M. Rastegari, and M. Farajtabar, "Relu strikes back: Exploiting activation sparsity in large language models," *arXiv preprint arXiv:2310.04564*, 2023.

[7] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re *et al.*, "Deja vu: Contextual sparsity for efficient llms at inference time," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 137–22 176.

[8] Y. Song, Z. Mi, H. Xie, and H. Chen, "Powerinfer: Fast large language model serving with a consumer-grade gpu," *arXiv preprint arXiv:2312.12456*, 2023.

[9] C. Song, X. Han, Z. Zhang, S. Hu, X. Shi, K. Li, C. Chen, Z. Liu, G. Li, T. Yang, and M. Sun, "Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models," *arXiv preprint arXiv:2402.13516*, 2024.

[10] Y. Song, H. Xie, Z. Zhang, B. Wen, L. Ma, Z. Mi, and H. Chen, "Turbo sparse: Achieving llm sota performance with minimal activated parameters," *arXiv preprint arXiv:2406.05955*, 2024.

[11] J. Shin, H. Yang, and Y. Yi, "Sparseinfer: Training-free prediction of activation sparsity for fast llm inference," *arXiv preprint arXiv:2411.12692*, 2024.

[12] L. S. Karumbunathan, "Nvidia jetson agx orin series," 2022.

[13] G. Gerganov, "llama.cpp," https://github.com/ggerganov/llama.cpp, 2023, accessed: November 18th, 2024.

[14] C. Karl, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[15] J. Liu, P. Ponnusamy, T. Cai, H. Guo, Y. Kim, and B. Athiwaratkun, "Training-free activation sparsity in large language models," *arXiv preprint arXiv:2408.14690*, 2024.

[16] W. J. Youden, "Index for rating diagnostic tests," *Cancer*, vol. 3, pp. 32–35, 1950.

[17] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou *et al.*, "Challenging big-bench tasks and whether chain-of-thought can solve them," *arXiv preprint arXiv:2210.09261*, 2022.

[18] EleutherAI, "lm-evaluation-harness," https://github.com/EleutherAI/lm-evaluation-harness, 2024, accessed: September 23th, 2024.

GS-TG: 3D Gaussian Splatting Accelerator with Tile Grouping for Reducing Redundant Sorting while Preserving Rasterization Efficiency

Joongho Jo
Korea University
Seoul, Republic of Korea
jojoss1004@korea.ac.kr

Jongsun Park
Korea University
Seoul, Republic of Korea
jongsun@korea.ac.kr

***Abstract*—3D Gaussian Splatting (3D-GS) has emerged as a promising alternative to neural radiance fields (NeRF) as it offers high speed as well as high image quality in novel view synthesis. Despite these advancements, 3D-GS still struggles to meet the frames per second (FPS) demands of real-time applications. In this paper, we introduce GS-TG, a tile-grouping-based accelerator that enhances 3D-GS rendering speed by reducing redundant sorting operations and preserving rasterization efficiency. GS-TG addresses a critical trade-off issue in 3D-GS rendering: increasing the tile size effectively reduces redundant sorting operations, but it concurrently increases unnecessary rasterization computations. So, during sorting of the proposed approach, GS-TG groups small tiles (for making large tiles) to share sorting operations across tiles within each group, significantly reducing redundant computations. During rasterization, a bitmask assigned to each Gaussian identifies relevant small tiles, to enable efficient sharing of sorting results. Consequently, GS-TG enables sorting to be performed as if a large tile size is used by grouping tiles during the sorting stage, while allowing rasterization to proceed with the original small tiles by using bitmasks in the rasterization stage. GS-TG is a lossless method requiring no retraining or fine-tuning and it can be seamlessly integrated with previous 3D-GS optimization techniques. Experimental results show that GS-TG achieves an average speed-up of 1.54 times over state-of-the-art 3D-GS accelerators.**

Keywords—Gaussian Splatting, Rendering, Accelerator

I. Introduction

3D Gaussian Splatting (3D-GS) [1] is emerging as a promising alternative to neural radiance fields (NeRF) [2], [3], [11], [12], [13] in the field of novel view synthesis. While NeRF implicitly models a scene using neural networks and synthesizes images through neural network inference, 3D-GS explicitly models a scene using millions of 3D Gaussians with learnable properties, and synthesizes images through tile-based rendering. This approach avoids the time-consuming sampling process of NeRF and enables parallel processing for faster rendering. Although 3D-GS has achieved significant speed-ups over NeRF, it still faces challenges in achieving the frames per second (FPS) required by 3D vision applications, such as augmented reality (AR) and virtual reality (VR) devices. For instance, even with Nvidia's server-grade A6000 GPU, the original 3D-GS [1] achieves only 15-25 FPS depending on the scene when synthesizing images at a resolution of 3840 x 2160. This falls short of the 90-120 FPS required for real-time interaction in binocular displays with a resolution of 2x(2014x2208), as required by recent AR/VR devices like Meta Quest 3.

Many previous works aiming to enhance the rendering speed of 3D-GS have applied well-known compression techniques such as quantization [6], densification [5], vector grouping [4], and pruning [6]. While these techniques intuitively reduce computational load, retraining or fine-tuning processes are required to compensate for the degradation in rendering quality. In contrast, algorithmic optimization efforts that improve the efficiency of the rendering pipeline itself without sacrificing rendering quality have been relatively scarce. Recently, GSCore [7] propose techniques that skip unnecessary computations in the tile selection and rasterization stages by considering the shape of Gaussians, enhancing the efficiency of the rendering pipeline. FlashGS [8] presents a method that more precisely recognizes the size and shape of Gaussians for tile selection than GSCore, and it proposes a workflow that operates efficiently running on GPUs.

In the tile-based rendering of 3D-GS, sorting and rasterization operations are performed independently for each tile. So, it inevitably leads to redundant sorting computations and unnecessary rasterization for Gaussians that affect multiple tiles. Our analysis reveals an interesting trade-off: increasing the tile size reduces redundant preprocessing and sorting operations but increases unnecessary rasterization computations. Previous optimization algorithms for 3D-GS rendering, however, have selected the tile sizes based solely on empirical observations of the fastest rendering speeds without thoroughly analyzing this trade-off. Furthermore, in-depth analyses and solutions for addressing this trade-off within the 3D-GS rendering pipeline have never been explored.

In this paper, we introduce GS-TG, a tile-grouping-based accelerator designed to improve the 3D-GS rendering speed by reducing redundant sorting operations while maintaining rasterization efficiency. By grouping tiles, GS-TG shares sorting operations across tiles within each group, significantly reducing redundant computations. During rasterization, a bitmask assigned to each Gaussian identifies its relevant tiles, enabling efficient reuse of sorting results across tiles. One of the main advantages of GS-TG is that it enhances 3D-GS rendering speed by enabling large tile sizes for sorting and smaller tile sizes for rasterization. Additionally, GS-TG is a completely lossless technique, requiring no retraining or fine-tuning, and it can be seamlessly integrated with previous 3D-GS rendering optimization methods.

II. Background

In this section, we introduce the tile-based rendering technique used in 3D Gaussian Splatting (3D-GS) and explain how this method can be applied within the 3D-GS rendering process.

A. Tile-Based Rendering

Tile-based rendering [17] is an approach that divides the output image into multiple small tiles and performs rendering for each tile individually. This method first identifies the objects that affect the pixel colors of each tile and then proceeds with rasterization on a per-tile basis. Such an approach offers several advantages, including reducing memory bandwidth, supporting parallel processing, and decreasing rasterization operations per pixel.

B. 3D Gaussian Splatting

3D Gaussian Splatting (3D-GS) [1] is an emerging technique in the field of novel view synthesis that explicitly models a scene using millions of learnable Gaussians and synthesizes images through tile-based rendering. Fig. 1 illustrates the overall rendering pipeline of 3D-GS. This pipeline consists of three primary stages executed sequentially: preprocessing, tile-wise sorting, and tile-wise rasterization.

In the **preprocessing stage**, as shown on the left side of Fig. 1, inputs such as the center position of 3D Gaussians (*3D_XYZ*), 3D

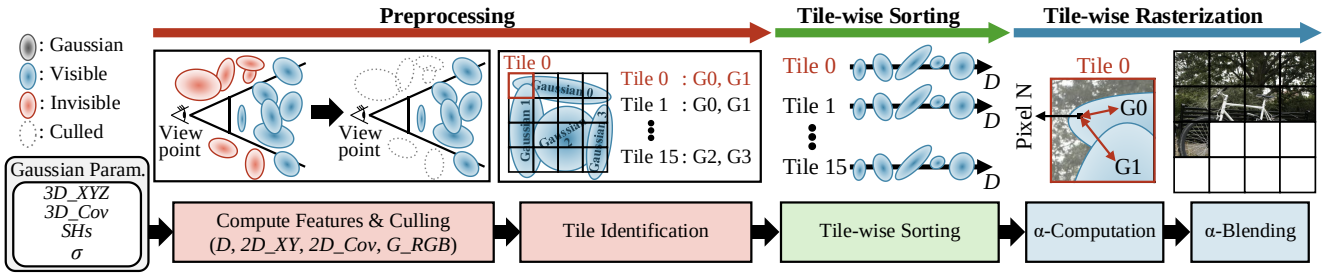


Fig. 1: The overall rendering pipeline of 3D Gaussian Splatting (3D-GS).

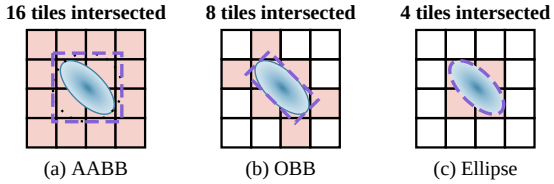


Fig. 2: Comparison of methods for calculating Gaussian influence based on size and shape: (a) AABB, (b) OBB, and (c) Ellipse boundary.

covariance matrix (*3D_Cov*), opacity (σ), and spherical harmonics coefficients (*SHs*) representing color across different viewpoints are utilized. During this stage, features required for culling and subsequent operations, including depth (*D*), 2D coordinates (*2D_XY*), 2D covariance (*2D_Cov*), and Gaussian color (*G_RGB*), are computed. Concurrently, invisible Gaussians for the current viewpoint are identified and excluded from rendering computations through a culling process. As illustrated in Fig. 1, the initial input includes both visible (blue) and invisible (red) Gaussians, but through culling operations, only Gaussians visible to the viewpoint are used in subsequent computations. Tile identification step, which identifies the Gaussians affecting each tile, uses *2D_XY* and *2D_Cov* to determine the tiles influenced by each Gaussian. While theoretically, the influence range of a Gaussian is infinite, the 3-sigma rule is applied to define the Gaussian's size in [1]. In the **tile-wise sorting stage**, for each tile, the Gaussians identified during preprocessing are sorted in the order of proximity to the viewpoint by utilizing their depth values (*D*). Finally, in the **tile-wise rasterization stage**, the sorted sequence of Gaussians for each tile is utilized to compute pixel colors in the order of proximity to the current viewpoint. As shown in Fig.1, rasterization consists of two main steps. The first step, known as α-computation, uses *2D_XY*, *2D_Cov*, and σ to calculate how much each Gaussian influences the color of the pixel being processed. Then, α-computation for i-th Gaussian is calculated using the following equation:

$$\alpha_i = \sigma_i * \exp\left(-\frac{1}{2}(P - 2D_XY_i)^T 2D_Cov_i^{-1}(P - 2D_XY_i)\right), \quad (1)$$

where P means the pixel's coordinates, and $2D_Cov_i^{-1}$ is the inverse of i-th Gaussian's 2D covariance matrix. In [1], when α_i is less than 1/255, i-th Gaussian is considered to have no influence on the pixel color and is therefore excluded from the subsequent α-blending computation. In α-blending step, pixel colors are computed starting from the one closest to the viewpoint, using the following equation:

$$Pixel\ Color = \sum_{i=1}^{N} G_RGB_i \alpha_i \prod_{k=1}^{i-1}(1 - \alpha_k), \quad (2)$$

where N is the number of visible Gaussian associated with the pixel. During the accumulation operation in (2), an early-exit is triggered if $\prod_{k=1}^{i-1}(1 - \alpha_k)$ falls below a predefined threshold (e.g., 10^{-4} as suggested in [1]).

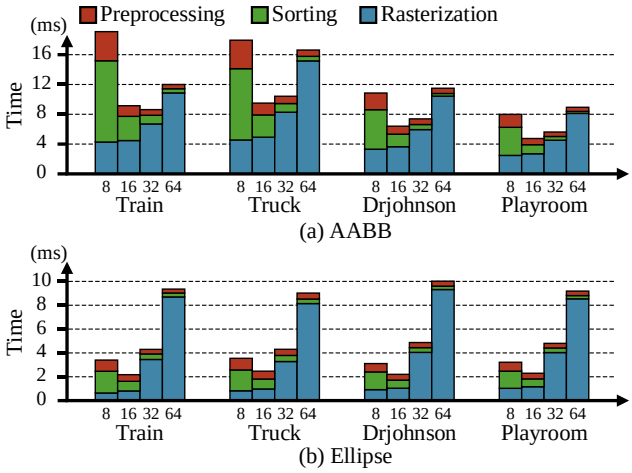


Fig. 3: Runtime breakdown across tile sizes for four scenes at different boundaries: (a) AABB and (b) Ellipse boundary.

C. Related Works

Recent advances, such as GSCore [7] and FlashGS [8], have improved 3D-GS rendering by introducing techniques to accurately identify the tiles influenced by Gaussians. These methods consider the size and shape of each Gaussian, reducing the number of Gaussians required for tile-wise sorting and rasterization. Fig. 2 illustrates three methods for calculating the influence of Gaussians. Fig. 2 (a) depicts an example of axis-aligned bounding boxes (AABB), where the Gaussian intersects with 16 tiles when using AABB. This approach, adopted in the original 3D-GS, is computationally efficient but fails to account for the shape of the Gaussian, resulting in significant unnecessary computations. Fig. 2 (b) demonstrates the use of oriented bounding boxes (OBB), where the Gaussian is identified to intersect with 8 tiles. This method, employed in GSCore, reduces unnecessary tile identification compared to AABB but comes at a relatively higher computational cost. Fig. 2 (c) presents the ellipse-based boundary, where the Gaussian intersects with only 4 tiles. This approach, adopted in FlashGS, minimizes unnecessary tile selection, but it is computationally more expensive as it directly handles the elliptical shape rather than a rectangular boundary. However, these works primarily focus on considering the size and shape of Gaussians, without conducting an in-depth analysis of the impact of *tile size* across the stages of the rendering pipeline. In the following section, we present the importance of *tile size* across the stages of the rendering pipeline.

III. Motivation

In this section, we first profile the 3D-GS rendering pipeline under various tile sizes. Subsequently, the impact of tile size on each stage of the 3D-GS rendering pipeline is analyzed.



Fig. 4: Tile identification examples for (a) large and (b) small tile sizes.

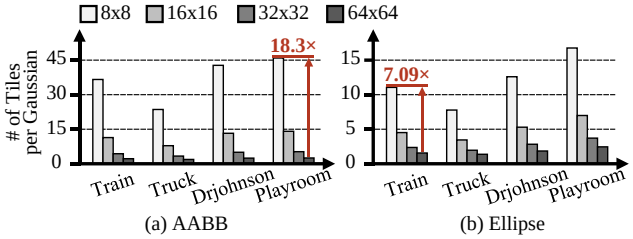


Fig. 5: Average number of intersecting tiles per Gaussian for various tile sizes in (a) AABB and (b) Ellipse boundary across four scenes.

TABLE I. THE PERCENTAGE OF GAUSSIANS SHARED WITH ADJACENT TILES BASED ON TILE SIZE

%	8×8	16×16	32×32	64×64
Train	94.4	89.0	79.7	66.0
Truck	89.0	79.2	64.7	47.7
Drjohnson	91.4	83.9	71.3	54.0
Playroom	91.3	83.8	71.7	54.7
Average	91.5	84.0	71.9	55.6

A. *Runtime Trade-offs in 3D-GS Rendering Across Different Tile Sizes*

To analyze the rendering performance of 3D-GS with different tile sizes, profiling is conducted on the NVIDIA A6000. Fig. 3 presents the runtime breakdown across four scenes—Train, Truck, Drjohnson, and Playroom—using various tile sizes (8×8, 16×16, 32×32, and 64×64). The results are shown separately for AABB in Fig. 3(a) and ellipse boundary in Fig. 3(b). For the analysis, we use and modify the author-released codes from [1] and [8]. Detailed information on the scenes and pretrained models can be found in section VI-A. As shown in Fig. 3, the runtime for the preprocessing and sorting stages decreases with larger tile sizes, whereas the rasterization stage finishes more quickly with smaller tile sizes. Generally, a tile size of 16×16 provides the fastest rendering speed, though in some cases, 32×32 can also be faster. However, previous optimization algorithms for 3D-GS rendering have selected tile sizes based solely on empirical observations of rendering speed, without a comprehensive analysis of this underlying trade-off. In section III-B, we conduct an in-depth analysis of this trade-off to identify the opportunities for accelerating 3D-GS rendering.

B. *Analysis of the Impact of Tile Size at Each Stage*

Preprocessing stage. In the preprocessing stage, tile size affects the computations in the 'tile identification' step, which identifies the tiles influenced by each Gaussian. Fig. 4 illustrates the examples of the tile identification process for (a) a large tile size and (b) a small tile size. As shown in the figure, reducing the tile size increases the number of tiles across the same image area, thereby increasing the number of tiles that each Gaussian needs to process during the tile identification step. Consequently, as the tile size decreases, the runtime of the preprocessing stage increases.

Tile-wise sorting stage. In Fig. 4 (a), Gaussian 0 intersects with 7 tiles and shares 2 of these tiles (highlighted with a red border) with Gaussian 1. Since sorting is performed on a per-tile basis, Gaussian 0 undergoes 7 sorting operations, 2 of which are redundant with Gaussian 1. In Fig. 4 (b), Gaussian 0 intersects with 21 tiles, sharing 4 tiles with Gaussian 1, resulting in redundant sorting operations. This outcome is inevitable, as reducing tile size increases the number of tiles within the same image size. Fig. 5 shows the average number of intersecting tiles per Gaussian under the same conditions as in Fig. 3. The results demonstrate that as the tile size decreases, the number of intersecting tiles per Gaussian increases exponentially. For the playroom scene at AABB, with an 8×8 tile size, the average number of tiles intersected by each Gaussian is 18.3 times higher than with a 64×64 tile size. Table I presents the percentage of Gaussians shared with adjacent tiles across different tile sizes. Notably, with an 8×8 tile size, an average of 91.5% of Gaussians are shared with adjacent tiles. In the tile-wise sorting stage, our analysis indicates that in 3D-GS rendering, as the tile size decreases, the number of sorting operations per tile increases exponentially, making a significant portion of these sorting operations redundant.

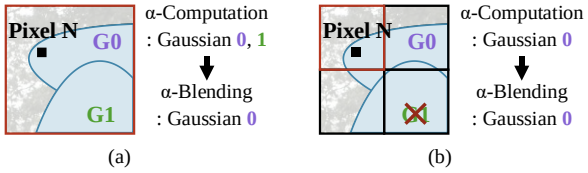


Fig. 6: Tile-wise rasterization examples for (a) large and (b) small tile sizes.

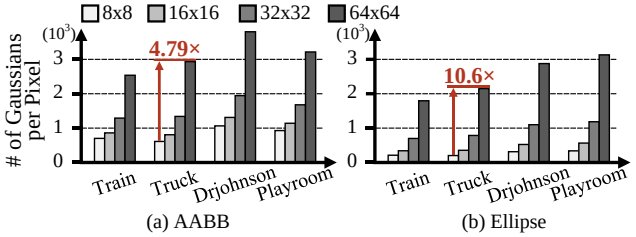


Fig. 7: Average number of Gaussians that need to be processed per pixel in (a) AABB and (b) Ellipse across four scenes.

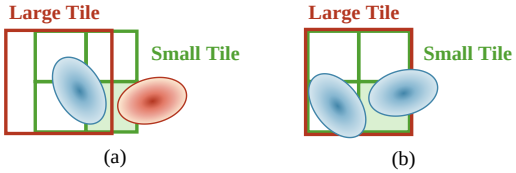


Fig. 8: Scenarios of small tile alignment within a large tile: (a) Misaligned and (b) Perfectly aligned.

Tile-wise rasterization stage. Fig. 6 illustrates an example of rasterization with (a) a large tile size and (b) a small tile size. In Fig. 6 (a), the tile containing Pixel N intersects with both Gaussian 0 and Gaussian 1, resulting in rasterization operations for Pixel N with respect to both Gaussians. In this case, Gaussian 0 influences Pixel N, necessitating both α-computation and α-blending. However, since Gaussian 1 does not affect Pixel N, an α-computation is performed to determine its influence, concluding that Gaussian 1 does not impact Pixel N, thereby excluding it from the α-blending operation. In Fig. 6 (b), only Gaussian 0 intersects the tile containing Pixel N, so unlike in the large tile size case, no α-computation is required for Gaussian 1. As this example shows, reducing the tile size allows for finer identification of Gaussian influence during the preprocessing stage, reducing unnecessary computations in the rasterization stage. Fig. 7 presents the average number of Gaussians that need to be processed per pixel under the same conditions as Fig. 3. The graph reveals that across all scenes and boundaries, as tile size increases, so does the number of Gaussians processed per pixel, indicating an increase in unnecessary α-computation operations with larger tile sizes. For the truck scene at ellipse with a 64×64 tile size, the average number of Gaussians that need to be processed per pixel is 10.6 times higher than with an 8×8 tile size. In the tile-wise rasterization operations, our analysis indicates that larger tile sizes lead to an increase in unnecessary rasterization computations.

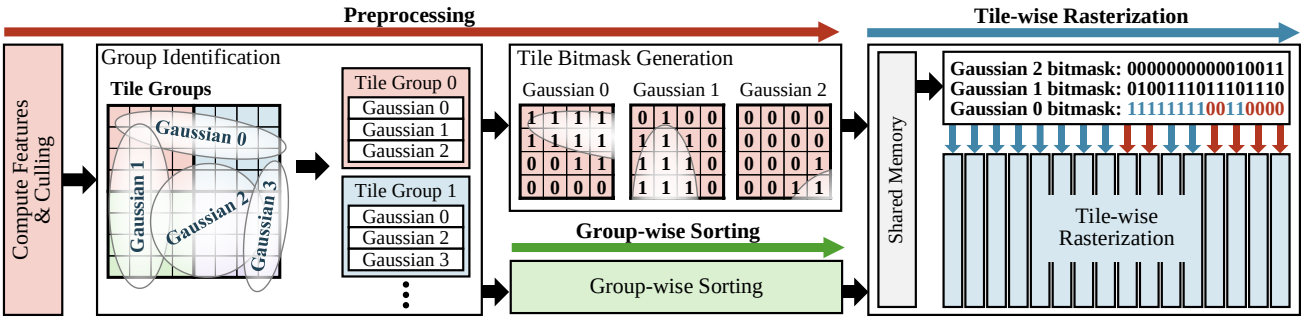


Fig. 9: the overall rendering pipeline of GS-TG.

IV. Proposed Tile-Grouping-Based Rendering Pipeline

A. Challenges in Balancing Tile Sizes for Efficient 3D-GS Rendering

Our analysis shows that larger tiles reduce preprocessing and sorting runtime, while smaller tiles improve rasterization efficiency. However, achieving both goals presents two challenges. First, using both small and large tiles requires tile identification operations for both tile sizes during the preprocessing stage, which leads to computational overhead. Second, a method is required to utilize data sorted at the larger tile size for rasterization at the smaller tile size. To address these challenges, we introduce GS-TG, tile-grouping-based rendering pipeline that enhances 3D-GS rendering speed by reducing redundant sorting operations while preserving rasterization efficiency.

B. Overall Process of GS-TG

To efficiently address both challenges with minimal overhead, smaller tiles should completely fit within larger tiles. Fig. 8 illustrates two scenarios: (a) when smaller tiles do not perfectly align with the larger tile, and (b) when they do. As shown in Figure 8(a), when smaller tiles are not perfectly aligned with the larger tile, some Gaussians (e.g., the red Gaussian in Fig. 8 (a)) may affect only the smaller tiles and not the larger tile. By contrast, as shown in Fig. 8 (b), when smaller tiles are perfectly aligned with the larger tile, all Gaussians that affect the smaller tiles also affect the larger tile. In this way, when smaller tiles fit perfectly within the larger tile, computational independence of the larger tile is ensured. Specifically, once the Gaussians affecting the larger tile are identified, all rendering operations for the smaller tiles within the larger tile can be performed using only those Gaussians. This computational independence provides three key advantages: 1) Even when rendering the smaller tiles using only the Gaussians that affect the larger tile, lossless results are guaranteed. 2) The operation of identifying the Gaussians affecting the smaller tiles and the sorting operation for the larger tile can be performed independently. 3) Gaussians affecting the smaller tiles can be selected using a simple bitmask, eliminating the need for large index overhead. As a result, ensuring that smaller tiles fit perfectly within the larger tile effectively resolves the aforementioned challenges. Therefore, we propose a tile-grouping-based rendering method that treats groups of tiles as if they are larger tiles. This ensures the computational independence of the tile groups (larger tiles).

Fig. 9 illustrates the overall pipeline of tile grouping-based rendering. First, feature calculations and culling operations are performed similarly to conventional 3D-GS rendering.

Group Identification Step. Tiles are grouped based on predefined group size, and the Gaussians influencing each group are identified. This step is similar to the tile identification step in conventional 3D-GS rendering but operates at the group level, which corresponds to larger tiles. For instance, in Fig. 9, 16 small tiles are grouped together, and the Gaussians influencing each group are determined. By grouping tiles, we ensure that all the small tiles within a group are influenced only by the Gaussians associated with that group. This relationship allows group-wise sorting and simultaneous bitmask generation to determine which Gaussians influence specific tiles within a group. The runtime overhead of identifying Gaussian influence at the small tile level is effectively hidden by these parallel operations. Furthermore, group-level sorting substantially mitigates redundant computational overhead by minimizing repetitive sorting operations, which are otherwise inherent in small tile-level sorting approaches.

Bitmask Generation Step. Once groups are identified, bitmasks are generated to enable tile-wise rasterization. For each Gaussian, a 16-bit bitmask is used to indicate which tiles within a group are influenced. For example, in Fig. 9, bitmasks are created for the Gaussians in Group 0 to identify which of the 16 small tiles are influenced. A tile is marked as "1" in the bitmask if it is affected by the Gaussian. Due to the deterministic relationship wherein tiles within a group are exclusively influenced by the corresponding Gaussians, a 16-bit bitmask efficiently encodes these influences, thereby optimizing both memory representation and computational complexity. This compact encoding substantially reduces the overhead involved in managing Gaussian-tile relationships at smaller tile sizes.

Tile-wise Rasterization Step. Using the group-sorted data and generated bitmasks, rasterization is performed at the smaller tile level. This minimizes unnecessary α-calculations by limiting rasterization to tiles explicitly marked in the bitmask.

For the computation of Gaussian influence during group identification and tile bitmask generation, any of the following techniques can be applied: AABB, OBB, or ellipse boundary. Fig. 9 demonstrates an example where the ellipse technique is applied.

V. Hardware Architecture

A. Motivation

The proposed GS-TG technique is designed to simultaneously perform bitmask generation and group-wise sorting. So, it enables preprocessing and sorting at the large tile group level while rasterization occurs at the small tile level. However, the single instruction, multiple threads (SIMT) architecture of GPUs presents limitations in parallel execution of bitmask generation and group-wise sorting. As a result, it becomes challenging to hide the execution time of the computations related to the impact of Gaussians on small tiles during the preprocessing stage. To overcome these limitations, a specialized hardware architecture has been designed to efficiently support the proposed rendering pipeline.

B. Architecture Overview

Fig. 10 illustrates the overall hardware architecture and key block diagram of GS-TG. The architecture consists of two primary modules: preprocessing module (PM) and GS-TG Core, both of which are implemented in four parallel instances to support efficient processing. PM performs Gaussian feature calculation, culling, and group identification operations while maintaining the same structure as the preprocessing stage in conventional 3D-GS rendering. Culled Gaussian features are passed to the GS-TG Core for subsequent stages in the GS-TG rendering pipeline. The GS-TG Core comprises three main hardware modules: bitmask generation module (BGM), group-wise sorting module (GSM), and rasterization module (RM).
BGM. This module determines the tiles influenced by each Gaussian

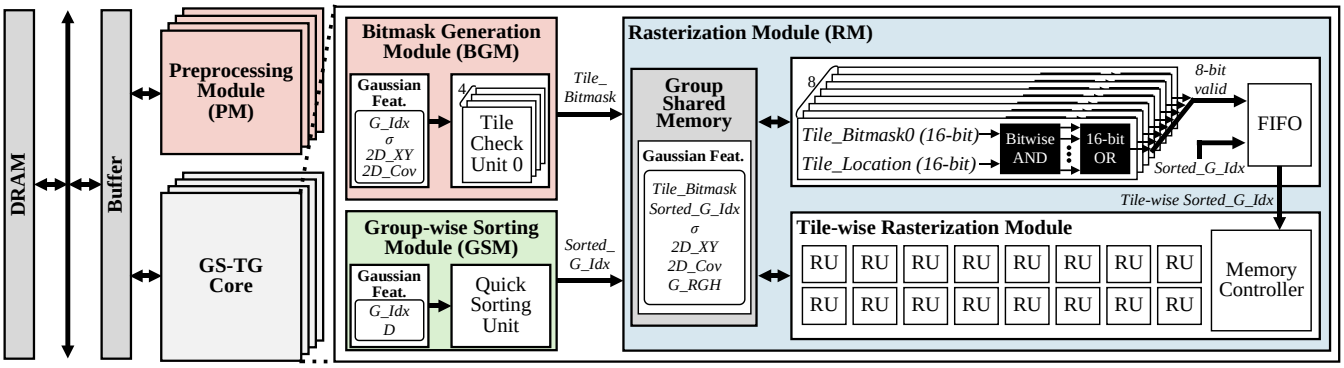


Fig. 10: The overall hardware architecture of GS-TG and the detailed block diagram of each module.

using opacity (σ), projected 2D coordinates (*2D_XY*), and covariance (*2D_Cov*) data. Four tile check units operate in parallel to generate a 16-bit tile bitmask (*Tile_Bitmask*), which is sent to the RM. **GSM.** Using the depth information (*D*) of each Gaussian, this module performs quick sorting. The sorting unit is equipped with 16 comparators, enabling high-speed sorting. The sorted Gaussian indices (*Sorted_G_Idx*) are forwarded to the RM. **RM.** This module is central to efficiently processing tile-based rendering, and it performs the following key operations: 1) Utilizing the 16-bit *Tile_Bitmask* and 16-bit *Tile_Location*, it applies bitwise AND operations to filter Gaussians affecting specific tiles. It also applies bitwise OR operations to generate valid flags, indicating whether a Gaussian impacts the current tile. This process handles eight Gaussians in parallel. 2) Gaussian indices identified for specific tiles are stored in a first-in-first-out (FIFO) buffer to maintain rendering order, and those are subsequently transferred to tile-specific rasterization submodules. Tile-wise rasterization module is composed of 16 parallel rasterization units (RUs). Each RU performs α-computation and α-blending to calculate final pixel colors, which are then output to memory.

This ensures seamless and efficient parallel processing across all stages, maximizing the performance of the GS-TG rendering pipeline.

VI. Experimental Results

A. Methodology

Algorithm. As mentioned in Section V-A, due to the single instruction multiple thread (SIMT) architecture of GPUs, it is challenging to execute bitmask generation and group-wise sorting concurrently in parallel. Therefore, GS-TG is designed to operate sequentially in the following order: group identification, tile bitmask generation, group-wise sorting, and tile-wise rasterization—whether running on a GPU. GS-TG is implemented by modifying the author-released codes from [8] using CUDA programming. Experiments are conducted on an NVIDIA A6000 GPU, and the codes are compiled with GCC 11.4.0 and NVCC in CUDA 11.6. **Hardware.** The proposed GS-TG architecture is implemented in RTL and synthesized using Synopsys Design Compiler with 28nm CMOS technology. Power and energy consumption are simulated using Synopsys PrimeTime PX, and speed improvements are evaluated using a cycle-level simulator. The DRAM bandwidth was set to 51.2 GB/s, and DRAM energy consumption is calculated based on [16]. **Common Details.** Table II presents the resolution and types of datasets used for evaluation. From the Tanks&Temples [9], Deep Blending [10], Mill-19 [14], and UrbanScene3D [15] datasets, various scenes representing indoor and outdoor real-world environments are selected: train, truck, Drjohnson, playroom, rubble, and residence. Simulations are performed using pre-trained 3D-GS-30k models. And, to improve the throughput and area efficiency of GS-TG, the models trained in 32-bit floating point are converted to 16-bit floating point. Following the methodology proposed in Mip-NeRF360, a train/test split is applied to all datasets. For the T&T and DB datasets, every 8th image is used for testing. In the Mill-19 dataset, every 64th image is used for testing, while in the UrbanScene3D dataset, every 128th image is used for testing.

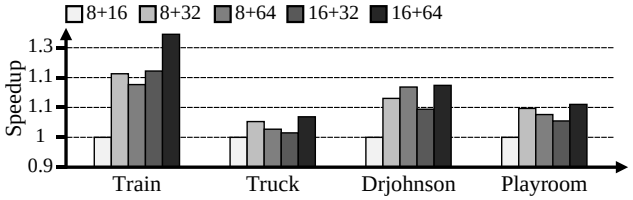


Fig. 11: Speedup of GS-TG with various boundary method combinations across four scenes running on GPU.

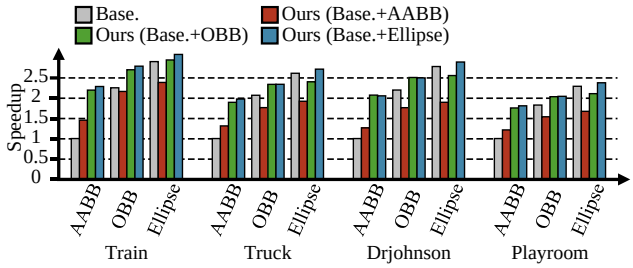


Fig. 12: Speedup of GS-TG with various boundary method combinations across four scenes running on GPU.

TABLE II. **Resolution and Types of Datasets Used for Evaluation**

Dataset	Scene	Resolution	Type
Tanks&Temples [9]	Train Truck	1959×1090 1957×1091	Outdoor
Deep Blending [10]	Drjohnson Playroom	1332×876 1264×832	Indoor
Mill-19 [14]	Rubble	4608×3456	Outdoor
UrbanScene3D [15]	Residence	5472×3648	Outdoor

B. Algorithm Evaluation

We first evaluate the performance of various tile sizes and group sizes by considering the parallel operations of bitmask generation and group-wise sorting. Fig. 11 shows the normalized speedup across four scenes (train, truck, Drjohnson, playroom). In Fig. 11, "8+16" indicates a tile size of 8×8 and a group size of 16×16, meaning that each group consists of 4 tiles. According to our analysis, the combination of 16+64 yielded the fastest performance in most cases. Therefore, the subsequent experiments with the proposed GS-TG are conducted with a tile size of 16×16 and a group size of 64×64.

We evaluate the speedup of GS-TG running on a GPU. Fig. 12 illustrates the performance improvements achieved on four scenes: train, truck, Drjohnson, and playroom. The x-axis labels AABB, OBB, and Ellipse represent the boundary methods used in the baseline tile identification step to calculate the influence of

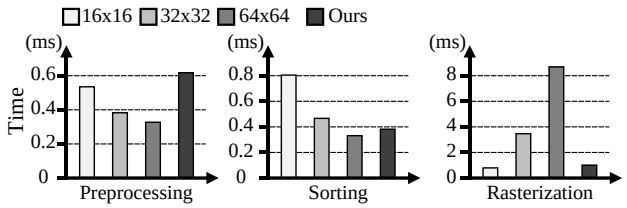


Fig. 13: Stage-wise runtime breakdown of the rendering pipeline for the Train scene, comparing the baseline approach with GS-TG running on GPU.

Gaussians. The speed-up results are normalized to the baseline case with AABB applied. In the figure, 'ours' represents the cases where the boundary methods shown on the x-axis are applied in the group identification step instead of the baseline's tile identification step. The red, green, and blue colors represent the cases where AABB, OBB, and Ellipse boundary methods are applied, respectively, in the bitmask generation step of GS-TG. For example, the green bar for AABB indicates the case where AABB is used in the group identification step, while OBB is applied in the bitmask generation step. From Fig. 12, three key findings are observed: 1) When using the Ellipse+Ellipse combination in GS-TG, performance surpassed all the baselines. 2) When the same boundary is applied to both the baseline and GS-TG, GS-TG consistently achieves faster speeds. For instance, GS-TG with OBB+OBB outperformed the baseline using OBB. 3) The proposed tile grouping-based rendering technique demonstrated flexible compatibility with various boundary methods.

Fig. 13 illustrates the stage-wise runtime breakdown of the three rendering pipeline stages (preprocessing, sorting, and rasterization) for the train scene. It compares the baseline, which uses the Ellipse boundary approach with tile sizes of 16×16, 32×32, and 64×64, to the proposed GS-TG employing Ellipse+Ellipse boundary. As shown in the figure, the proposed GS-TG achieves sorting performance comparable to a 64×64 tile size by grouping 16×16 tiles into 16 groups during the sorting stage, while maintaining rasterization performance equivalent to a 16×16 tile size. However, when GS-TG runs on a GPU, the bitmask generation operations cannot overlap with the sorting operations during runtime, causing the preprocessing stage to be slower than the baseline. This issue has been resolved through the dedicated GS-TG accelerator, enabling parallel execution of bitmask generation and group sorting, thereby further enhancing the overall performance.

C. Hardware Evaluation

Table III presents the synthesis results for each module of GS-TG. The total area of GS-TG is 3.984 mm^2, with a power consumption of 1.063W and an operating frequency of 1 GHz. To the best of our knowledge, the only existing hardware related to 3D-GS rendering is GSCore [7]. Therefore, we set the baseline as the conventional 3D-GS rendering using the Ellipse boundary running on the proposed accelerator and compared the performance of GS-TG to GSCore. Fig. 14 illustrates the speedup achieved by GS-TG across six different scenes compared to the baseline and GSCore. GS-TG achieves a geometric mean speedup of 1.33 times over the baseline, with a maximum speedup of 1.58 times in the high-resolution residence scene. Additionally, GS-TG outperforms GSCore by up to 1.54 times in the residence scene. Fig. 15 presents the energy efficiency achieved by GS-TG compared to the baseline and GSCore across six distinct scenes. GS-TG achieves a geometric mean energy efficiency improvement of 2.12 times over the baseline, with a maximum improvement of 2.97 times observed in the high-resolution Residence scene. This superior performance is attributed to the specialized architecture optimized for the tile-grouping-based rendering pipeline.

VII. Conclusion

In this paper, we thoroughly investigated the impact of tile size on the rendering pipeline of 3D Gaussian splatting (3D-GS) and identified key bottlenecks. To address these challenges, we introduced GS-TG, a tile-grouping-based rendering method that

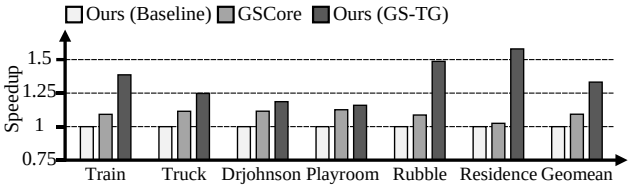


Fig. 14: Normalized speedup of GS-TG across six different scenes, compared to the baseline and GSCore.

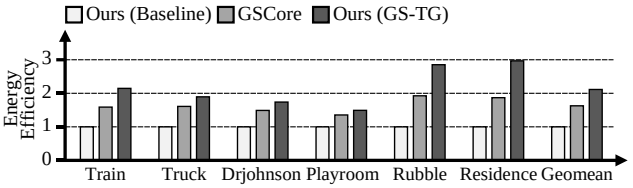


Fig. 15: Normalized energy efficiency of GS-TG across six different scenes, compared to the baseline and GSCore.

TABLE III. **Hardware configuration**

Module		Configuration	Area [mm^2]	Power [W]
PM		4	0.648	0.429
GS-TG Core	BGM	4	0.051	0.055
	GSM	4	0.012	0.001
	RM	4	1.891	0.338
Buffer		4×2×42KB	1.382	0.240
Total			3.984	1.063
Operating Freq.			1GHz	

significantly enhances the speed of 3D-GS rendering by reducing redundant sorting operations and optimizing rasterization efficiency. GS-TG is a lossless approach, requiring no retraining or fine-tuning, and can be seamlessly integrated with existing 3D-GS optimization techniques. Experimental results show that GS-TG consistently outperforms state-of-the-art 3D-GS hardware, achieving up to 1.54 times speed-up. This work highlights the potential for further hardware-software co-design optimizations in 3D-GS rendering.

Acknowledgment

This work was supported in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2024-00345481); in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2024-00405495, Plug&Play(P&P) Chiplet Integration research center); in part by the Ministry of Trade, Industry and Energy(MOTIE) and Korea Institute for Advancement of Technology(KIAT) through the "International Cooperative R&D program"(Task No. P0028486)

REFERENCES

[1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering.," ACM Trans. Graph., vol. 42, no. 4, pp. 139–1, 2023.

[2] B. Mildenhall et al, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis", In ECCV, 2020.

[3] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," ACM Trans. Graph., vol. 41, no. 4, pp. 1–15, Jul. 2022.

[4] S. Niedermayr, J. Stumpfegger, and R. Westermann, "Compressed 3d gaussian splatting for accelerated novel view synthesis," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 10349–10358.

[5] S. Kim, K. Lee, and Y. Lee, "Color-cued Efficient Densification Method for 3D Gaussian Splatting," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 775–783.

[6] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS," Mar. 29, 2024, arXiv.

[7] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim, "GSCore: Efficient Radiance Field Rendering via Architectural Support for 3D Gaussian Splatting," in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 3, La Jolla CA USA: ACM, Apr. 2024, pp. 497–511.

[8] Guofeng Feng, Siyan Chen, Rong Fu, Zimu Liao, Yi Wang, Tao Liu, Zhilin Pei, Hengjie Li, Xingcheng Zhang, and Bo Dai. "FlashGS: Efficient 3d Gaussian Splatting for Large-Scale and High-resolution Rendering", Aug. 15, 2024, arXiv.

[9] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: benchmarking large-scale scene reconstruction," ACM Trans. Graph., vol. 36, no. 4, pp. 1–13, Aug. 2017.

[10] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," ACM Trans. Graph., vol. 37, no. 6, pp. 1–15, Dec. 2018.

[11] Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "TensoRF: Tensorial Radiance Fields," in ECCV, 2022.

[12] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 5501–5510.

[13] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," Advances in Neural Information Processing Systems, vol. 33, pp. 15651–15663, 2020.

[14] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *CVPR*, 2022.

[15] Liqiang Lin, Yilin Liu, Yue Hu, Xingguang Yan, Ke Xie, and Hui Huang. Capturing, reconstructing, and simulating: the urbanscene3d dataset. In *ECCV*, 2022.

[16] Z. Zhou, J. Liu, Z. Gu, and G. Sun, "Energon: Toward efficient acceleration of transformers using dynamic sparse attention," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 1, pp. 136–149, 2022.

[17] M. Terai, J. Fujiki, R. Tsuruno, and K. Tomimatsu, "Tile-Based Modeling and Rendering," in Smart Graphics, vol. 4569.

AdreamDCO: AI-Driven Robust and Efficient Design Automation for Digitally Controlled Oscillators

Yaolong Hu[*$1], Hao Guo[*$2], Shikai Wang[#3], Jiaqi Liu[$4], Weidong Cao[#5], Taiyun Chi[$6]
[*]*Equally-Credited Authors (ECAs)* [$]*ECE Department of Rice University*, Houston, USA
[#]*ECE Department of The George Washington University*, Washington, D.C., USA
{[1]yh72, [2]hg52, [4]jl355, [6]Taiyun.Chi}@rice.edu, [3]shikai.wang@gwu.edu, [5]weidong.cao@email.gwu.edu

***Abstract*—This paper presents how we leverage AI-human collaboration to develop an end-to-end, automated design flow for digitally controlled oscillators (DCOs), a key radio-frequency (RF) integrated circuits (ICs) building block that dominates phase noise and jitter performance of RF systems. Specifically, we decompose the DCO design process into two steps and use AI to enhance productivity and optimize performance within each step. Additionally, we demonstrate how AI can assist RF IC designers in creating unconventional circuit components to tackle challenging design specifications. Overall, the proposed flow is capable of synthesizing the DCO design including the schematic and layout in 80 seconds after one-time training, and is frequency agile between 1 and 20 GHz. Moreover, it can select the most robust design under process variations when multiple design parameters meet target specifications under the nominal condition. The proposed automated DCO design flow is demonstrated using two silicon prototypes implemented in the GlobalFoundries 22-nm CMOS SOI process. In the measurements, they achieve >192.4-dBc/Hz figure-of-merit (FoM) and <1.5-kHz frequency resolution at 7.1 to 8.6 GHz and 3.8 to 4.6 GHz, outperforming existing manual designs at similar frequencies.**

***Index Terms*—artificial intelligence (AI), Bayesian optimization (BO), CMOS, design automation, differential evolution (DE), DCO, frequency tuning, RF IC, residual neural networks.**

I. Introduction

RF ICs are foundational to wireless communication and sensing systems, enabling applications such as Wi-Fi connectivity, cellular communications, automotive radars, and many more. Similar to analog ICs, the existing RF IC design flow is mostly manual and heavily dependent on human expertise. As the demand for higher performance and increased complexity in RF systems continues to grow, the conventional manual design flow poses significant challenges in meeting the target product-to-market time and performance specifications. Additionally, the current manual design flow involves extensive electromagnetic (EM) and nonlinear circuit simulations, both of which are computationally intensive, leading to long simulation times. Reliability issues, such as process variations, further complicate RF IC design and verification. Consequently, RF IC design automation is emerging as an active area of research [1]–[8].

Several recent works that leveraged AI to automate the design of passive networks within RF ICs have been reported [9]–[18]. While these tools can greatly reduce the design time for passive networks, it is important to note that the design of active circuits (i.e., building block topology, transistor sizing, and biasing) is equally crucial as the design of passive networks for RF ICs. Even after the initial passive network is ready, its integration with active components still requires extensive nonlinear circuit simulations and further optimizations to account for transistor nonidealities (such as parasitics and nonlinearity). This highlights the need for an "end-to-end" design automation flow that can jointly optimize active and passive components to streamline the overall RF IC design process.

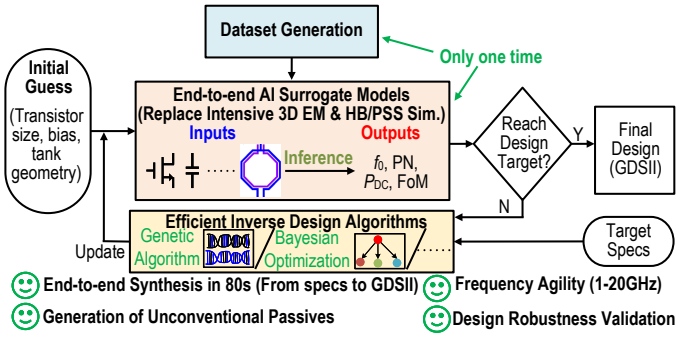


Fig. 1. Our proposed AI-assisted, automated DCO design flow and its key features.

To address this unmet need, this paper presents an AI-assisted end-to-end design flow and uses an LC-based digitally controlled oscillator (DCO) as a demonstration example. The LC-based DCO is one of the most critical building blocks in modern wireless and wireline transceivers, and its performance dominates the overall phase noise and jitter of the system clock. As shown in Fig. 1, our AI-assisted design flow consists of three steps: (i) generating datasets (as a one-time effort); (ii) training high-fidelity, end-to-end AI surrogate models to replace computationally intensive 3-D EM and nonlinear circuit simulations, enabling rapid evaluation of DCO performance based on active and passive design parameters; and (iii) developing efficient inverse design algorithms to automatically find the optimal design.

Our major contributions are summarized as follows:

- **End-to-end Synthesis for Active and Passive Co-**

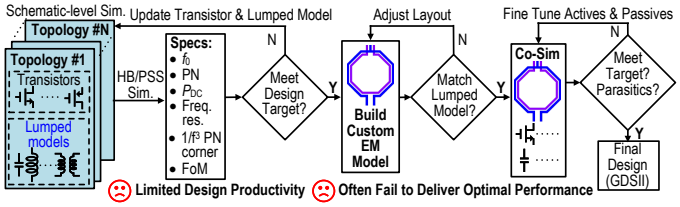


Fig. 2. Conventional manual DCO design flow.

Design: Once trained, the GDSII of the DCO tank, along with design parameters of the DCO transistor core (with the consideration of layout parasitics), is directly synthesized from target specifications within 80 seconds.

- **Generation of Unconventional Passives:** We demonstrate that the automated flow is capable of assisting human designers in creating new, unseen passive networks that outperform existing expert designs.
- **Frequency Agility:** The proposed flow can produce designs across a wide frequency range, anywhere between 1 and 20 GHz.
- **Design Robustness Validation:** The AI model enables rapid process variation simulations of the DCO tank, a task that is challenging using conventional EM solvers.

II. Background

A. Conventional, Manual DCO Design Flow

The conventional design flow of DCOs is both time-consuming and labor-intensive. As shown in Fig. 2, it begins with selecting a suitable DCO topology based on target specifications, a process that often requires deep design expertise. Next, designers build the DCO schematic and perform schematic-level simulations. In this step, lumped-element models are usually constructed to model the passive components in the circuit [e.g., inductors and transformers (XFMRs) [19]–[21]]. Despite their limited accuracy compared with scattering parameters (S-parameters) derived from 3-D EM simulations, these lumped-element models are crucial in manual designs, as they provide valuable intuitions for human designers and can guide subsequent optimizations.

Next, the transistor design parameters and lumped-element models are iteratively tuned to meet target DCO specifications. Typical specifications include oscillation frequency (f_0), phase noise (PN), DC power consumption (P_{DC}), frequency resolution, $1/f^3$ phase noise corner, and figure-of-merit (FoM), as shown in Fig. 2. Among these, the phase noise and $1/f^3$ phase noise corner are related to the frequency stability of the DCO output, which serves as the clock signal of the RF system. In this work, we monitor both $\mathrm{PN}_{10\mathrm{MHz}}$ to evaluate far-end phase noise at 10-MHz offset and $\mathrm{PN}_{10\mathrm{kHz}}$ to assess close-in phase noise at 10-kHz offset. A smaller difference between $\mathrm{PN}_{10\mathrm{kHz}}$ and $\mathrm{PN}_{10\mathrm{MHz}}$ indicates a better (lower) $1/f^3$ phase noise corner. Additionally, a low P_{DC} is highly desired, especially for battery-powered devices to extend battery life. Fine DCO frequency resolution is also preferred in communication systems for precise channel selection. Finally, the FoM is a comprehensive benchmark that takes f_0, phase noise, and P_{DC} into consideration. DCO design always aims to maximize FoM while ensuring all other specifications meet the system requirements.

After the schematic-level design, 3-D EM models of the passive components are created. Designers manually adjust the geometric parameters of the EM models, run EM simulations, and extract circuit parameters from the simulation results. This step is repeated until the extracted circuit parameters match those of the lumped-element model.

Following this step, the EM-simulated S-parameters of the passive components are imported into the circuit design environment for co-simulation with the active components. Given that RF IC performance is highly sensitive to device parasitics and nonlinearity, designers typically need to fine-tune the geometry of the 3-D EM models again and jointly optimize them with the active part until the overall DCO achieves the target performance.

This iterative and heuristic process is not only time-consuming but also prone to sub-optimal results, as the large design space increases the risk of being trapped in local optima. Moreover, discrepancies often arise between the lumped-element model and EM-simulated S-parameters of the passive components, especially for wide-band designs. Consequently, design productivity and overall performance are often limited by this manual design process.

B. Recent Advances in RF IC Design Automation

A few works have recently been reported that leveraged AI surrogate models to automate the design of *passive networks* within RF ICs. These models include convolutional neural networks (CNN) [9], multilayer perceptrons (MLP) [10], recurrent neural networks (RNN) [11], graph transformers [12], UNets [13], and graph neural networks (GNN) [14]. They are capable of predicting either the S-parameters of passive networks [9], [11]–[14] or their electrical parameters [10], [15]–[17], such as input impedance and loss, presented to active components. Beyond template-based synthesis, AI models have also been applied to non-heuristic passive network designs, such as the pixelated network in [18] and the XFMR with localized deformation in [13]. However, as mentioned earlier, *what is currently missing is the co-design of active and passive components in an end-to-end manner*. This gap is specifically addressed in our proposed design flow.

III. Proposed AI-assisted DCO Design Flow

A. Decomposing DCO Design Into Two Steps

As demonstrated in recent advancements in AI-assisted analog and RF IC design automation [2], [10], [14], [16], [18], adopting "AI-human collaboration" represents the optimal strategy, as it offers explainable and trustworthy designs while harnessing the full potential of AI. More importantly, *it allows circuit designers to focus on establishing circuit insights and exploring new circuit ideas without being bogged down by nuances.*

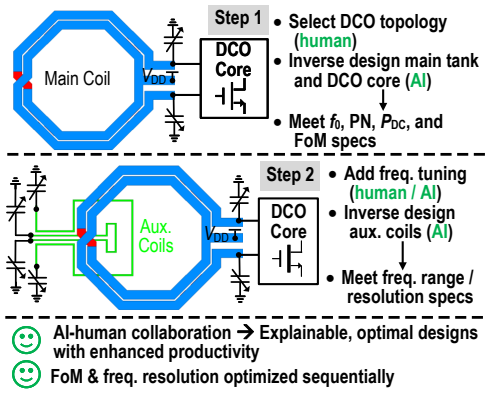


Fig. 3. Our proposed two-step DCO design automation flow.

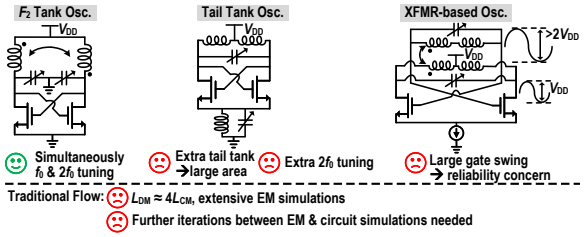


Fig. 4. Comparison between different main-tank topologies of the oscillator.

As shown in Fig. 3, we leverage this AI-human collaboration to guide our development and decompose the DCO design process into two steps. In the first step, we design the main tank and DCO transistor core to achieve the target phase noise and FoM. The second step then focuses on optimizing frequency tuning to meet the desired frequency resolution. The importance of this two-step approach is that the two most critical DCO metrics, i.e., FoM and frequency resolution, can be optimized sequentially. This significantly simplifies our development efforts, making it possible to establish an efficient automated design flow.

Within each step, we either rely on human expertise to select an existing topology (such as in *step 1*) or collaborate AI to develop a new topology (such as in *step 2*). Additionally, AI is employed in both steps to assist designers in enhancing design productivity and optimizing circuit performance.

B. Step 1: DCO Main Tank Design

There are three dominant oscillator topologies that present high FoM (see Fig. 4): the Class-F_2 tank topology [22]–[24], the tail-tank topology [25], [26], and XFMR-based topology [27], [28]. In this work, we selected the Class-F_2 topology for the design of the main tank and DCO core, as it achieves simultaneous fundamental (f_0) and second harmonic ($2f_0$) tuning without the need for a dedicated tail tank, while ensuring more reliable operation compared with the XFMR-feedback topology. However, the Class-F_2 tank requires the L_{DM} (differential-mode inductance) / L_{CM} (common-mode inductance) ratio to be $\approx$ 4 for simultaneous f_0 and $2f_0$ tuning [22]. In the traditional manual design flow, optimizing

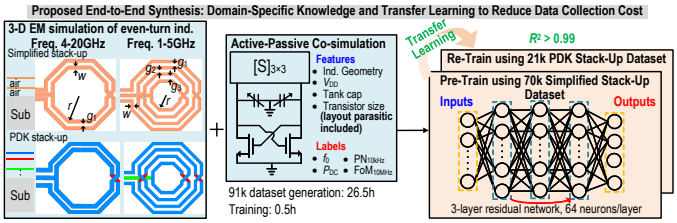
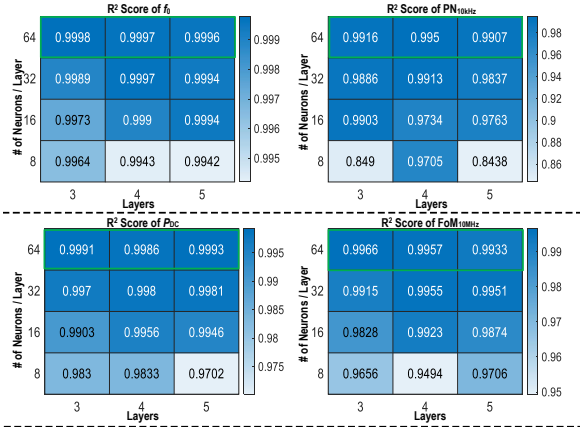


Fig. 5. Dataset generation and end-to-end AI surrogate model training of DCO main tank design.



Loss Function	R^2 Score of f_0	R^2 Score of PN_{10kHz}	R^2 Score of P_{DC}	R^2 Score of FoM_{10MHz}
MAPE (Mean Absolute Pct. Error)	-3.7214	0.5470	-0.0076	0.5119
MSE (Mean Squared Error)	0.9964	0.9871	0.9984	0.9895
MAE (Mean Absolute Error)	0.9998	0.9916	0.9991	0.9966
Huber (δ = 1.0)	0.9832	0.9664	0.9962	0.9816

Fig. 6. Comparison between different NN hyperparameters and loss functions.

such an inductor requires extensive EM simulations. Even after the initial inductor layout is ready, further iterations between passive and active co-optimizations are still needed to achieve the best possible FoM.

Our automated design flow addresses these challenges in an end-to-end manner, eliminating the need to explicitly consider the L_{DM} / L_{CM} ratio as an intermediate step. To reduce the data collection overhead associated with training high-fidelity models, we leveraged domain-specific knowledge and transfer learning (TL) [18]. For example, we focused on inductors with an even number of turns (2-turn and 4-turn inductors in this work to cover 1 to 20 GHz), as they naturally offer a smaller L_{CM} compared with inductors with an odd number of turns [22], making it easier to satisfy the desired L_{DM} / L_{CM} $\approx$ 4 relationship. Additionally, we first collected datasets using a simplified stack-up to train an initial model, which was then fine-tuned with a smaller number of samples collected using the complete PDK-specific stack-up, as shown in Fig. 5. This TL approach significantly reduces the dataset collection time by $\sim 3\times$. Once the S-parameters of the inductors were obtained from EM simulations, an end-to-end dataset was generated by co-simulating the S-parameters and the DCO core in Cadence Virtuoso. The overall data collection took $\sim$26.5 hours on a 128-core workstation, generating $\sim$91k data points.

We used the residual network architecture to build our AI

TABLE I
DESIGN EXAMPLES COVERING 1 TO 20 GHZ

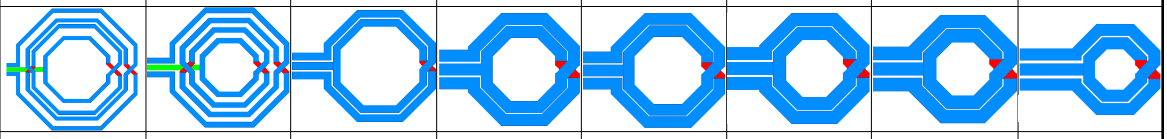
Frequency	1.6 GHz	4.0 GHz	8.0 GHz	10.0 GHz	12.0 GHz	14.0 GHz	16.0 GHz	20.0 GHz
Main Tank Layout								
Transistor Size	200µm/100nm	40µm/100nm	40µm/100nm	40µm/100nm	20µm/100nm	40µm/100nm	40µm/100nm	60µm/100nm
SE Cap	1400 fF	702 fF	654 fF	443 fF	257 fF	226 fF	207 fF	205 fF
$\text{FoM}_{10\text{MHz}}$	193.5 dBc/Hz	193.1 dBc/Hz	193.5 dBc/Hz	193.4 dBc/Hz	194.0 dBc/Hz	192.5 dBc/Hz	192.1 dBc/Hz	189.6 dBc/Hz
P_{DC}	0.78 mW	0.51 mW	0.66 mW	0.65 mW	0.38 mW	0.66 mW	0.71 mW	1.14 mW

TABLE II
COMPARISON BETWEEN DE AND BO

Algorithm	DE	BO
SE Capacitor Value (fF)	654	992
Inductor Radius (μm)	80	56
Inductor Coil Gap (μm)	3	14
Inductor Coil Width (μm)	14	8
Transistor Size	40μm / 100nm	40μm / 100nm
f_0 (GHz)	7.9996	8.0820
$\text{PN}_{10\text{kHz}}$ (dBc/Hz)	-73.9282	-59.0346
P_{DC} (mW)	0.664	0.827
$\text{FoM}_{10\text{MHz}}$ (dBc/Hz)	193.51	192.39
Elapsed Time (seconds)	36.9	33.5

surrogate model, due to its superior accuracy and scalability [15], [29]. During training, we experimented with different hyperparameters and loss functions, as shown in Fig. 6. We eventually chose 3 layers, 64 neurons per layer, and Mean Absolute Error (MAE) as the loss function, balancing between training overhead and accuracy. The model is lightweight, thanks to the domain-specific knowledge and transfer learning that were applied during data collection. The predicted R^2 score exceeds 0.99.

Once the surrogate model was ready for performance evaluation, we solved the specifications-to-parameters (S2P) inverse design problem using optimizations. Specifically, the inverse design problem was formulated as

$$\begin{aligned} \arg\min_{\Phi} \quad & -\text{FoM}_{10\text{MHz}} \\ \text{s.t.} \quad & |f_0 - f_{\text{target}}| = 0, \quad \text{PN}_{10\text{kHz}} < \text{PN}_{10\text{kHz,target}}, \\ & P_{\text{DC}} < P_{\text{DC,target}} \end{aligned} \tag{1}$$

where Φ represents the design space for both active and passive components, and $\text{FoM}_{10\text{MHz}}$ is the optimization object. The constraints include f_0, $\text{PN}_{10\text{kHz}}$, and P_{DC}.

A penalty function $\Psi(\Phi)$ was introduced, as

$$\begin{aligned} \Psi(\Phi) = & -\omega_1 \times \text{FoM}_{10\text{MHz}} + \omega_2 \times |f_0 - f_{\text{target}}| \\ & + \omega_3 \times elu(\text{PN}_{10\text{kHz}} - \text{PN}_{10\text{kHz,target}}) \\ & + \omega_4 \times elu(P_{\text{DC}} - P_{\text{DC,target}}) \end{aligned} \tag{2}$$

where $\omega_{1,2,3,4}$ are weights assigned to respective specifications, and the exponential linear unit (elu) penalizes constraint violations [30].

It is important to note that in the DCO design, the cross-coupled active core must provide sufficient loop gain to sustain oscillation. To exclude design parameters that fail to sustain oscillations, we trained an additional three-layer neural network to pre-screen design parameters, and any design failing to oscillate was assigned a large penalty.

We explored two methods to solve this inverse design problem: the differential evolution (DE) algorithm and Bayesian optimization (BO). To compare the performance between the two, we ran an experiment with the following design targets: $f_0 = 8.0$ GHz, $\text{PN}_{10\text{kHz}} < -75$ dBc/Hz, and $P_{DC} < 0.7$ mW. As shown in Table II, both DE and BO generated designs within 40 seconds. Additionally, the design synthesized by DE outperformed that by BO, particularly in terms of f_0 accuracy, $\text{PN}_{10\text{kHz}}$, P_{DC}, and $\text{FoM}_{10\text{MHz}}$. This advantage arises because DE directly utilizes the AI surrogate model to predict DCO performance, enabling nearly instantaneous performance evaluations. In contrast, BO relies on a Gaussian process regressor (GPR) to predict the objective function output. The GPR is iteratively updated using data points generated by the AI surrogate model, resulting in slower evaluations, reduced accuracy, and less effective convergence. Based on this experiment, we chose DE to solve the rest of the inverse design problems in this work.

Our AI surrogate model and inverse design are applicable anywhere between 1 and 20 GHz. Table I summarizes the inverse design results for the main tank and its associated single-ended (SE) capacitance and transistor size at a few frequencies across the 1–20 GHz range. All these designs achieved FoM of >189 dBc/Hz. Additionally, their performance predicted by the surrogate model closely matches the results obtained from EM and nonlinear circuit simulations, demonstrating the effectiveness of our approach.

C. Step 2: Adding Frequency Tuning Mechanism to the Main Tank

As for the second step of the DCO design, we demonstrate how we leverage AI to assist with the design of a new EM structure for fine frequency tuning. In this work, we set a challenging frequency resolution target of sub-1.5 kHz. This requires a switched capacitor array (SCA) with a total of 21 bits, assuming a typical 20% tuning range with margin. To enable design automation, we chose to use existing PDK capacitors instead of manually designing custom capacitors as in [26], [28], as this avoids the extensive overhead associated

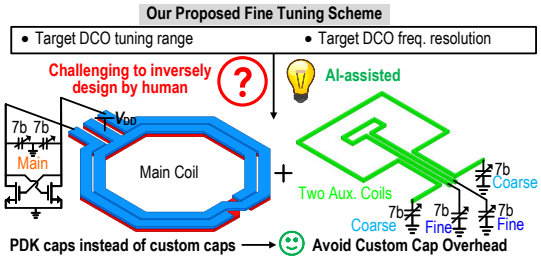


Fig. 7. Our proposed fine frequency tuning scheme incorporating an unconventional EM structure co-developed by AI.

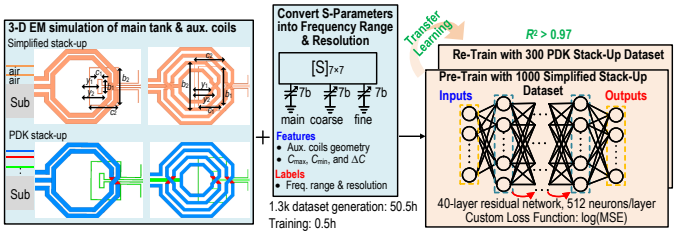


Fig. 8. End-to-end surrogate model for fine frequency tuning.

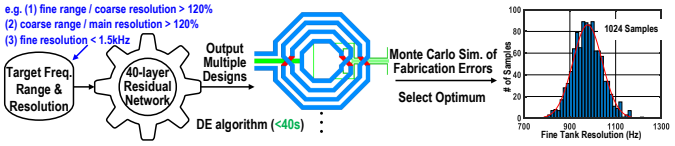


Fig. 9. AI-assisted design automation to identify the optimal layout of the auxiliary coils, achieving the target sub-1.5-kHz frequency resolution.

with manual layout, parasitic extraction, EM simulations, fine tuning, and potential mismatches between custom capacitor models and actual measurements.

Past work has shown that when an auxiliary coil is loaded with a capacitor and weakly coupled to the main tank, the effective capacitance presented to the main tank is significantly smaller [28], [31]. However, a single auxiliary coil is insufficient to meet our stringent frequency resolution target of sub-1.5 kHz, as the minimum capacitor required for a single auxiliary coil is still impractically small. As shown in Fig. 7, our design intuition suggests that introducing one more auxiliary coil, i.e., two auxiliary coils in total, with different magnetic couplings to the main tank could address this problem. While this high-level circuit idea is promising, deriving closed-form equations to guide the design is overly complex. Moreover, in the traditional manual design flow, we must rely on a lumped-element model with numerous mutually coupled parameters for this new EM structure to complete the initial schematic-level design. Optimizing this lumped-element model to achieve the desired frequency tuning range and resolution is itself challenging, let alone finding an EM structure that aligns the lumped-element model. All these intricate design and layout nuances necessitate collaboration with AI.

As shown in Fig. 8, similar to the *Step 1*, we began by generating datasets with different geometric parameters of the

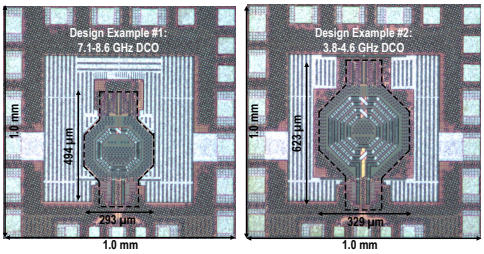


Fig. 10. Die micrographs of the two DCO designs implemented in the GlobalFoundries 22-nm CMOS SOI process.

auxiliary coils and recording their S-parameters from EM simulations. These S-parameters were then co-simulated with the main-coil SCA, coarse-tuning SCA, and fine-tuning SCA to derive the frequency tuning range and resolution for each SCA: f_{TR1} and f_{Res1} for the main coil, f_{TR2} and f_{Res2} for the coarse-tuning coil, and f_{TR3} and f_{Res3} for the fine-tuning coil. The $f_{\mathrm{TR1,2,3}}$ and $f_{\mathrm{Res1,2,3}}$ are further saved as "labels" for training.

We applied the same TL approach as in Section III-B to reduce data collection overhead. A total of 1,000 passive structures were simulated using a simplified metal stack, and an additional 300 samples were simulated with the PDK-specific metal stack. Next, a 40-layer AI surrogate model based on the residual architecture was trained. The gradients of traditional MSE and MAE tend to diminish when their values are very small, which significantly slows down the convergence. To address this issue, we introduced a logarithmic term in the loss function, as

$$loss = \log_{10}[\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2] \tag{3}$$

where Y_i is the ground-truth label, and $\hat{Y}_i$ is the predicted value. This loss function enabled faster convergence, and the model achieved an R^2 score of >0.97 during testing.

Finally, the inverse design was performed using DE. Similar to Section III-B, the inverse design problem was formulated as an optimization problem, as

$$\arg\min_{\Xi} \quad f_{\mathrm{Res3}}, \quad \text{s.t.} \quad \frac{f_{\mathrm{TR2}}}{f_{\mathrm{Res1}}} > \delta_1, \frac{f_{\mathrm{TR3}}}{f_{\mathrm{Res2}}} > \delta_2 \tag{4}$$

where Ξ represents the layout parameters of the auxiliary coils, and the constraints are to ensure that the tuning range of the coarse (or fine) SCA is at least $\delta_1\times$ (or $\delta_2\times$) the frequency resolution of the main (or coarse) SCA to avoid frequency tuning "dead zones" over PVT variations.

During the optimization, we targeted a sub-1.5-kHz frequency resolution with δ_1 and δ_2 set to 120%. The DE algorithm efficiently produced multiple layouts that met the target resolution and design constraints in <40 seconds. To select the optimal layout, we evaluated how the frequency resolution was affected by potential fabrication errors, as on-chip passive components are prone to inaccuracies caused by the lithography, such as variations in line width and coil position. As shown in Fig. 9, leveraging the AI surrogate

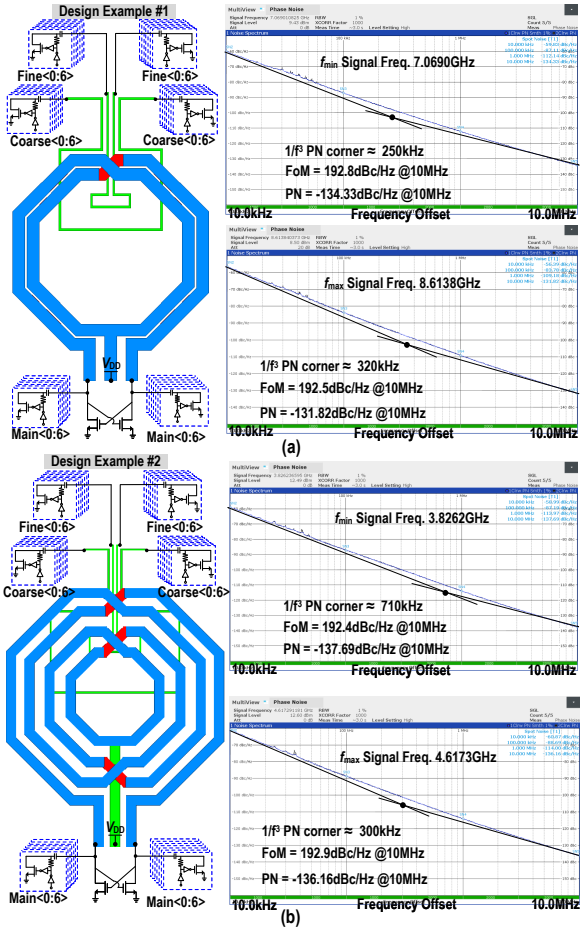


Fig. 11. (a) Schematic of design example #1 with a two-turn inductor as the main tank, along with measured phase noise plots at the minimum and maximum frequencies. (b) Design example #2 with a four-turn inductor as the main tank. The main, coarse, and fine frequency tunings for both designs are realized using three 7-bit SCAs with existing PDK capacitors.

model, we were able to complete 1,024 Monte Carlo simulations in just 15 seconds on a personal laptop, compared with ∼90 hours using EM simulations on a 64-core workstation. This capability of quickly predicting the impact of process variations is particularly valuable for high-volume production.

IV. Experiment Results

The proposed AI-assisted DCO design automation was demonstrated using two examples, taped out using the GlobalFoundries 22-nm CMOS SOI process. The die micrographs are shown in Fig. 10.

The first design targets emerging beyond-5G communications in the upper mid-band with a tuning range of 7.1 to 8.6 GHz and employs a two-turn inductor as its main tank [see the schematic in Fig. 11(a)]. The measured frequency tuning resolution is 0.225 to 1.5 kHz, the $1/f^3$ phase noise corner is 250 to 320 kHz, and the $\text{FoM}_{10\text{MHz}}$ is 192.5 to 193.0 dBc/Hz across the whole tuning range. The measured phase noise plots are also shown in Fig. 11(a).

The second design targets a major 5G communication band, the n77 band, with a tuning range of 3.8 to 4.6 GHz. It utilizes a four-turn inductor as its main tank [see the schematic in Fig. 11(b)]. The measured frequency tuning resolution is 0.525 to 1.5 kHz, the $1/f^3$ phase noise corner is 300 to 710 kHz, and the $\text{FoM}_{10\text{MHz}}$ is 192.4 to 193.3 dBc/Hz across the tuning range.

TABLE III
Performance Comparison with State-of-the-Art DCOs at Similar Frequencies

Ref.	ISSCC 2017 [28]	ISSCC 2019 [32]	RFIC 2022 [26]	**This Work**	
Main Tank Topology	Trifilar-Coil	Folded-Stacked	Class-B	**Class-F_2**	
Freq. Tuning Mechanism	XFMR Feedback	SCA	SCA	**Weakly Coupled Aux Coils**	
Freq. (GHz)	3.2 to 4	4.15 to 4.97	8.2 to 10.2	**7.1 to 8.6**	**3.8 to 4.6**
Tuning Range	22%	18%	24%	**19.8%**	**18.8%**
Res. (kHz)	1.3	N/A	17.5	**<1.5**	
V_{DD} (V)	0.4	0.2	1.2	**0.4**	
$\text{PN}_{10\text{MHz}}$ (dBc/Hz)	-145 to -143	-144.3 to -142.2	-134* to -129*	**-134.3 to -131.8**	**-137.8 to -136.2**
$\text{FoM}_{10\text{MHz}}$ (dBc/Hz)	190	191 to 195.8	178.0* to 181.3*	**192.5 to 193.0**	**192.4 to 193.3**
$1/f^3$ PN Corner (kHz)	120	70 to 700	N/A	**250 to 320**	**300 to 710**
P_{DC} (mW)	3.8	1.77 to 2.28	13	**0.63 to 0.71**	**0.45 to 0.49**
Core Area (mm^2)	0.11	0.272	0.13	**0.11**	**0.16**
Custom Cap?	Yes	N/A	Yes	**No**	
Design Automation?	No	No	No	**Yes (AI-assisted end-to-end synthesis.)****	
Tech.	16-nm FinFET	22-nm FDSOI	28-nm CMOS	**22-nm FDSOI**	

*Graphically Estimated.
** Dataset collection: 77 hours, training: 1 hour, inverse design: 80 seconds.
$\text{FoM} = |\text{PN}| + 20\log_{10}(f_0/\Delta f) - 10\log_{10}(P_{\text{DC}}/1\text{mW})$

Compared with prior works at similar frequencies (see Table III), both designs achieve state-of-the-art performance, with >192.4-dBc/Hz $\text{FoM}_{10\text{MHz}}$ and <1.5-kHz frequency resolution. It is important to note that such high performance is achieved in a compact, single-inductor footprint and without using any custom capacitors as in other manually designed DCOs [26], [28]. These results clearly demonstrate the effectiveness of our proposed DCO design automation flow and its capability to achieve state-of-the-art DCO performance that outperforms existing human expert designs.

V. Conclusion

In this paper, we present a two-step AI-assisted end-to-end DCO design automation flow and demonstrate its effectiveness using two silicon prototypes. Their measurement results highlight the unique advantages of our proposed design flow, including (i) significantly reducing design time from several weeks (per our own design experience) to ∼80 seconds (other than one-time 77 hours of data collection and 1 hour of training), (ii) achieving globally optimal designs, (iii) frequency agility (1-20 GHz), and (iv) generating new, unseen passive networks to achieve functions beyond expert designs. This automated design flow can be readily extended to other frequencies, processes, and building blocks, offering a promising solution to enhance RF IC design productivity.

REFERENCES

[1] W. Lyu *et al.*, "Multi-objective Bayesian Optimization for Analog/RF Circuit Synthesis," in *Proc. ACM/IEEE Design Autom. Conference (DAC)*, 2018, pp. 1–6.

[2] D. Krylov *et al.*, "Learning to Design Analog Circuits to Meet Threshold Specifications," in *Proc. Int. Conf. on Mach. Learn. (ICML)*, 2023, pp. 17 858–17 873.

[3] Z. Wang *et al.*, "An Automated and Process-Portable Generator for Phase-Locked Loop," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2021, pp. 511–516.

[4] Y. Hu *et al.*, "Automated mmWave Power Amplifier Design Flow and a 28-GHz Design Example in 45-nm CMOS SOI," in *Proc. IEEE/MTT-S Int. Microw. Symp. (IMS)*, 2024, pp. 842–845.

[5] W. Cao *et al.*, "Domain Knowledge-Infused Deep Learning for Automated Analog/Radio-Frequency Circuit Parameter Optimization," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, p. 1015–1020.

[6] J. Poojary *et al.*, "Exploration of Design / Layout Tradeoffs for RF Circuits using ALIGN," in *Proc. IEEE Radio Freq. Integr. Circuits Symp. (RFIC)*, 2023, pp. 57–60.

[7] M. Fayazi *et al.*, "FuNToM: Functional Modeling of RF Circuits Using a Neural Network Assisted Two-Port Analysis Method," in *Proc. ACM/IEEE Int. Conf. on Comput.-Aided Design (ICCAD)*, 2023, pp. 1–8.

[8] Z. Gao *et al.*, "Correlated Bayesian Model Fusion: Efficient High-Dimensional Performance Modeling of Analog/RF Integrated Circuits Over Multiple Corners," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 42, no. 2, pp. 360–370, Feb. 2023.

[9] S. Qi and C. D. Sarris, "Deep Neural Networks for Rapid Simulation of Planar Microwave Circuits Based on Their Layouts," *IEEE Trans. Microw. Theory Techn.*, vol. 70, no. 11, pp. 4805–4815, Nov. 2022.

[10] D. Lee *et al.*, "Neural-Network-Based Automated Synthesis of Transformer Matching Circuits for RF Amplifier Design," *IEEE Trans. Microw. Theory Techn.*, vol. 70, no. 11, pp. 4726–4739, Nov. 2022.

[11] O. Waqar Bhatti, N. Ambasana, and M. Swaminathan, "Design Space and Frequency Extrapolation: Using Neural Networks," *IEEE Microw. Mag.*, vol. 22, no. 10, pp. 22–36, Oct. 2021.

[12] D. Kim *et al.*, "TraceFormer: S-parameter Prediction Framework for PCB Traces based on Graph Transformer," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2024.

[13] H. Chae *et al.*, "PulseRF: Physics-Augmented ML Modeling and Synthesis for High-Frequency RFIC Design," in *Proc. ACM/IEEE Int. Conf. on Comput.-Aided Design (ICCAD)*, 2024, pp. 1–9.

[14] G. Zhang, H. He, and D. Katabi, "Circuit-GNN: Graph Neural Networks for Distributed Circuit Design," in *Proc. Int. Conf. on Mach. Learn. (ICML)*, 2019, pp. 7364–7373.

[15] D. Munzer *et al.*, "Residual Network Based Direct Synthesis of EM Structures: A Study on One-to-One Transformers," in *Proc. IEEE Radio Freq. Integr. Circuits Symp. (RFIC)*, 2020, pp. 143–146.

[16] S. Er *et al.*, "Deep Learning Assisted End-to-End Synthesis of mm-Wave Passive Networks with 3D EM Structures: A Study on A Transformer-Based Matching Network," in *Proc. IEEE/MTT-S Int. Microw. Symp. (IMS)*, 2021, pp. 66–69.

[17] F. Passos *et al.*, "PACOSYT: A Passive Component Synthesis Tool Based on Machine Learning and Tailored Modeling Strategies Towards Optimal RF and mm-Wave Circuit Designs," *IEEE J. Microw.*, vol. 3, no. 2, pp. 599–613, Apr. 2023.

[18] E. A. Karahan, Z. Liu, and K. Sengupta, "Deep-Learning-Based Inverse-Designed Millimeter-Wave Passives and Power Amplifiers," *IEEE J. of Solid-State Circuits*, vol. 58, no. 11, pp. 3074–3088, Nov. 2023.

[19] Y. Hu and T. Chi, "A 27.5–46.2-GHz Broadband Low Noise Amplifier With IP_3 Enhancement," *IEEE Microw. and Wireless Technol. Lett.*, vol. 33, no. 9, pp. 1293–1296, Sept. 2023.

[20] Y. Hu and T. Chi, "A 27–46-GHz Low-Noise Amplifier With Dual-Resonant Input Matching and a Transformer-Based Broadband Output Network," *IEEE Microw. and Wireless Compon. Lett.*, vol. 31, no. 6, pp. 725–728, June 2021.

[21] Y. Hu and T. Chi, "A Systematic Approach to Designing Broadband Millimeter-Wave Cascode Common-Source With Inductive Degeneration Low Noise Amplifiers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 4, pp. 1489–1502, Apr. 2023.

[22] M. Shahmohammadi, M. Babaie, and R. B. Staszewski, "25.4 A 1/f Noise Upconversion Reduction Technique Applied to Class-D and Class-F Oscillators," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2015, pp. 1–3.

[23] D. Murphy, H. Darabi, and H. Wu, "25.3 A VCO with Implicit Common-Mode Resonance," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2015, pp. 1–3.

[24] M. Shahmohammadi, M. Babaie, and R. B. Staszewski, "A 1/f Noise Upconversion Reduction Technique for Voltage-Biased RF CMOS Oscillators," *IEEE J. of Solid-State Circuits*, vol. 51, no. 11, pp. 2610–2624, Nov. 2016.

[25] A. Ismail and A. Abidi, "CMOS Differential LC Oscillator with Suppressed Up-Converted Flicker Noise," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2003, pp. 98–99 vol.1.

[26] L. Wang *et al.*, "An 8.2-10.2 GHz Digitally Controlled Oscillator in 28-nm CMOS Using Constantly-Conducting NMOS Biased Switchable Capacitor," in *Proc. IEEE Radio Freq. Integr. Circuits Symp. (RFIC)*, 2022, pp. 207–210.

[27] M. Babaie and R. B. Staszewski, "Third-Harmonic Injection Technique Applied to a 5.87-to-7.56GHz 65nm CMOS Class-F Oscillator with 192dBc/Hz FOM," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2013, pp. 348–349.

[28] C.-C. Li *et al.*, "19.6 A 0.2V Trifilar-Coil DCO with DC-DC Converter in 16nm FinFET CMOS with 188dB FOM, 1.3kHz Resolution, and Frequency Pushing of 38MHz/V for Energy Harvesting Applications," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2017, pp. 332–333.

[29] K. He *et al.*, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. on Comput. Vis. and Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

[30] J. Liu *et al.*, "From Specification to Silicon: Towards Analog/Mixed-Signal Design Automation using Surrogate NN Models with Transfer Learning," in *Proc. ACM/IEEE Int. Conf. On Comput.-Aided Design (ICCAD)*, 2021, pp. 1–9.

[31] W. Wu, J. R. Long, and R. B. Staszewski, "High-Resolution Millimeter-Wave Digitally Controlled Oscillators With Reconfigurable Passive Resonators," *IEEE J. of Solid-State Circuits*, vol. 48, no. 11, pp. 2785–2794, Nov. 2013.

[32] O. El-Aassar and G. M. Rebeiz, "26.5 A 0.1-to-0.2V Transformer-Based Switched-Mode Folded DCO in 22nm FDSOI With Active Step-Down Impedance Achieving 197dBc/Hz Peak FoM and 40MHz/V Frequency Pushing," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 416–418.

DCO-3D: Differentiable Congestion Optimization in 3D ICs

Hao-Hsiang Hsiao[1], Yi-Chen Lu[2], Pruek Vanna-iampikul[3], Anthony Agnesina[2], Rongjian Liang[2], Yuan-Hsiang Lu[1], Haoxing Ren[2], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, GA
[2]NVIDIA, Santa Clara, CA, USA; [3]Deparment of Electrical Engineering, Burapha University, Chonburi, Thailand;
{thsiao, yuan-hsiang.lu, limsk}@gatech.edu; {yilu, aagnesina, rliang, haoxingr}@nvidia.com;

***Abstract*—State-of-the-art 3D IC flows fail to consider 3D congestion during earlier stages, leading to excessive use of end-of-flow ECO resources for routability correction that severely degrades full-chip Power, Performance, and Area metrics. We present DCO-3D, a Machine Learning-based routability-aware 3D PD flow that performs early post-route congestion prediction using Siamese Networks and resolves the predicted hotspots using a fully differentiable 3D cell spreading with Graph Neural Network. On 6 industrial designs in a commercial 3nm node, DCO-3D improves Pin-3D, the known best Pin-3D flow, by up to 47.2% in overflow, 86.2% in TNS and 5.1% in power at signoff.**

I. Introduction

State-of-the-art (SOTA) 3D Physical Design (PD) flows leverage 2D commercial Place and Route (P&R) tools to build signoff-quality 3D Integrated Circuits (ICs), a methodology commonly referred to as the "pseudo" approach [1]. Despite significant advances in improving the Power, Performance, and Area (PPA) envelope, the challenge of routability in 3D ICs remains a critical and underexplored issue, posing a bottleneck to further advancements in 3D integration.

Decades of research in 2D PD have underscored the critical role of routability optimization during the placement stage in achieving superior full-chip PPA metrics [2], [3]. However, this foundational design principle has yet to be fully adopted in the context of 3D ICs. The advent of Machine Learning (ML)-based data-driven approaches has driven the development of various methodologies [4], [5], [6], [7], [8], [9] aimed at predicting routability in early PD stages, enabling proactive optimization. However, these advancements are limited to 2D ICs and cannot be seamlessly extended to 3D ICs due to the unique and inherent complexity of 3D designs.

To address these limitations, we propose DCO-3D, the first ML-driven, routability-aware 3D PD flow that incorporates 3D congestion prediction using customized Siamese networks [10]. Building on this prediction, we introduce a fully differentiable 3D congestion optimization framework that minimizes congestion while preserving placement quality. Unlike prior 2D approaches, our framework enables cell spreading across all three dimensions—x, y, and z—facilitating congestion redistribution not only horizontally and vertically but also across tiers. This cross-tier capability effectively utilizes 3D resources to resolve congestion hotspots that are intractable in conventional 2D layouts.

DCO-3D can seamlessly integrate into any existing 3D PD flow. In this work, we specifically compare it against Pin-3D [11], the widely recognized SOTA 3D design flow, to highlight its advantages. Figure 1 illustrates how our DCO-3D methodology (highlighted in red) integrates with the Pin-3D flow.

DCO-3D takes a 3D global placement as input and directly generates cell spreading decisions in TCL constraints for the commercial P&R tool, *Synopsys ICC2*. Notably, DCO-3D introduces no additional PD optimization steps beyond those already present in the Pin-3D flow. Instead, it provides supplemental TCL and Python scripts to guide cell spreading decisions for congestion optimization within the existing tool. Experimental results demonstrate that this early optimization yields significant full-chip PPA improvements by the end of the flow. This work represents the *first* routability-aware 3D PD flow, offering accurate and efficient congestion optimization during the early design stages. Our main contributions are as follows:

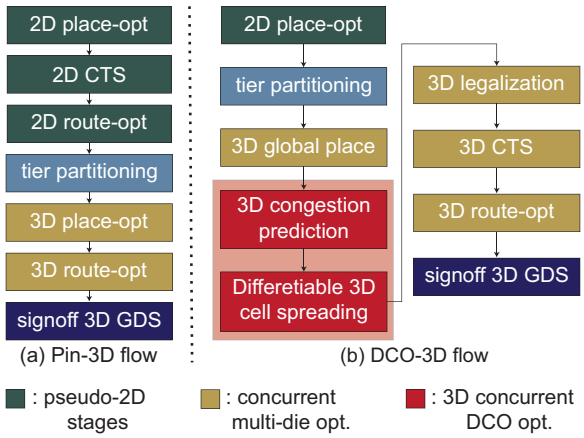


Fig. 1: SOTA 3D PD flow Pin-3D [11] vs. our ML-driven routability-aware 3D flow DCO-3D, which introduces additional differentiable 3D cell spreading for early and precise congestion optimization.

- We propose DCO-3D, the first commercial-quality 3D PD flow that performs congestion optimization for 3D ICs.
- We introduce the first congestion optimization framework that leverages the z-dimension, allowing cells to dynamically move between tiers to optimize congestion.
- We develop a fully differentiable, multi-objective cell spreading methodology using GNN, facilitating direct gradient-based optimization.
- We design a customized Siamese network tailored for concurrent multi-die 3D congestion prediction. Existing 2D routability prediction methods [4], [5], [6], [7], [8], [9] are inadequate for addressing such 3D concurrent challenges.
- We achieve significant 3D full-chip PPA improvements over the SOTA 3D PD flow Pin-3D [11], demonstrated on six industrial designs in a commercial $3nm$ technology node.

II. Preliminaries

A. State-of-the-art 3D Flow

Pseudo-3D [1], unlike academic true-3D approaches [12], [13], [14], [15], [16], utilizes commercial tools to construct 3D ICs, ensuring commercial PPA quality and manufacturing readiness in GDS format. The Pseudo-3D flow [17] strategically leverages commercial tools to determine the optimal (x, y) locations of standard cells while assigning z-coordinates through tier assignments. In this work, we adopt the SOTA Pin-3D [11] flow to directly optimize 3D placement for improved routability. As illustrated in Figure 1, the process

encompasses placement, Clock Tree Synthesis (CTS), post-CTS optimization, routing, and timing closure, all within a 3D context.

B. Early Congestion Estimation

1) RUDY (Rectangular Uniform wire DensitY)

RUDY[18] is routing demand estimation based solely on the pin locations. The RUDY of a net e can be computed as follows: For a net with bounding box coordinates $\{x_e^{\mathrm{l}}, y_e^{\mathrm{l}}, x_e^h, y_e^h\}$, its RUDY at any location (x, y) is proportional to the wire area divided by the bounding box area:

$$RUDY_e(x,y) \propto \mu_e \left(\frac{(w_e + h_e)}{w_e h_e} \right) \propto \mu_e \left(\frac{1}{w_e} + \frac{1}{h_e} \right) \quad (1)$$

where $w_e = x_e^{\mathrm{h}} - x_e^{\mathrm{l}}$ and $h_e = y_e^{\mathrm{h}} - y_e^{\mathrm{l}}$, and μ_e is an indicator function that checks whether (x, y) lies within the bounding box of net e. Given the design is divided into multiple tiles, the RUDY at tile (m, n) is:

$$RUDY_{(m,n)} = \sum_{e \in N} \left(\frac{1}{w_e} + \frac{1}{h_e} \right) \frac{\text{Area}_{tile \cap bbox}}{\text{Area}_{tile}} \quad (2)$$

2) PinRUDY

PinRUDY, an extension of RUDY for estimating pin density, is defined as follows:

$$\text{PinRUDY}(m,n) = \sum_{pin \in tile_{mn}} \left(\frac{1}{w_e} + \frac{1}{h_e} \right), \quad pin \in e \quad (3)$$

C. Overview of Our Approach

This study addresses two critical challenges in 3D IC design: (1) predicting post-3D routing congestion maps from a given 3D global placement ("images-to-images" prediction), and (2) refining placements based on these predictions to mitigate congestion hotspots while preserving signoff-quality QoR.

Our methodology consists of two main stages:

1) 3D Congestion Prediction: Predict post-route congestion maps to identify potential hotspots.
2) 3D Congestion Optimization: Utilize these predictions to perform 3D cell spreading and alleviate congestion.

III. ML-BASED ROUTABILITY PREDICTION

A. Dataset Construction

The proposed prediction framework relies on a prebuilt dataset for supervised learning. Using our in-house 3nm process design kit (PDK), we synthesize RTL into gate-level netlists with Synopsys Design Compiler, followed by Pseudo-3D P&R using *Synopsys IC Compiler II (ICC2)*. For each netlist, 300 diverse 3D placement layouts are generated by sampling placement parameters listed in Table I, serving as training input. Ground truth congestion maps are obtained by completing 3D CTS and routing for each layout.

B. Data Engineering

In this subsection, we describe the input features of the prediction model, the training labels, and the data processing methods.

1) Input Feature Maps

During global routing, the chip layout is divided into routing bins, referred to as GCells. Our approach partitions each die in a 3D placement based on these GCell coordinates to generate feature maps that aid in predicting the post-3D routing layout. Each pixel in these maps encapsulates specific local information corresponding to its respective bin area. We designed the following feature maps specifically for 3D ICs:

- Cell Density: The ratio of cell area within a bin to the bin's area.
- Pin Density: The number of pins per unit area
- 2D RUDY: Routing demand for 2D nets (nets with all pins on a single die), as described in Section II-B.

TABLE I: 3D Placement Parameters used for constructing placement dataset

Placement Parameter	type	value
coarse.pin_density_aware	bool	"false", "true"
coarse.target_routing_density	float	[0, 1]
coarse.adv_node_cong_max_util	float	[0, 1]
coarse.congestion_driven_max_util	float	[0, 1]
coarse.cong_restruct_effort	enum	[0, 4]
coarse.cong_restruct_iterations	int	[0, 10]
coarse.enhanced_low_power_effort	enum	[0, 4]
coarse.low_power_placement	bool	"false", "true"
coarse.max_density	float	[0, 1]
legalize.displacement_threshold	int	[0, 10]
initial_place.two_pass	bool	"false", "true"
initial_drc.global_route_based	bool	"false", "true"
flow.enable_ccd	bool	"false", "true"
initial_place.effort	enum	[0, 2]
final_place.effort	enum	[0, 2]
flow.enable_irap	bool	"false", "true"

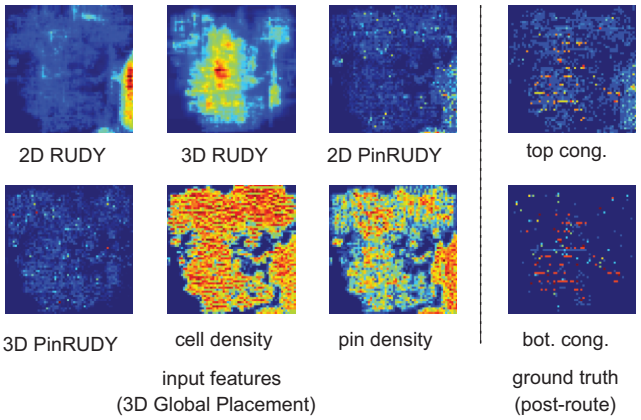


Fig. 2: Visualization of input features and ground truth: "2D" features represent nets with all pins located on the same die, whereas "3D" features capture nets with pins spanning across different dies.

- 3D RUDY: Routing demand for 3D nets (nets connecting pins across multiple dies), scaled by **0.5** to account for additional 3D routing resources.
- 2D PinRUDY: Pin-based routing demand II-B for 2D nets.
- 3D PinRUDY: Pin-based routing demand II-B for 3D nets.
- Macro Blockage: Area occupied by macros.

2) Training Label Maps

The training labels for each data point are congestion maps for both the top and bottom dies, as the objective is to predict these maps concurrently. Each congestion map is divided into bins corresponding to GCells, with each bin assigned a routing overflow value derived from the congestion report generated by the tool. A visualization of these maps from a selected data sample is presented in Figure 2.

3) Pre- and Post-Processing of Maps

The overall data processing pipeline is illustrated in Figure 3. For each design in the dataset, the placement layout is divided into bins based on GCell dimensions to compute the feature maps. Similarly, the post-route layout is divided into GCell bins to generate the label maps. To handle varying GCell grid sizes across different designs, both feature and label maps are resized to a fixed dimension of $H \times W$, where H and W denote the height and width required by the convolutional neural network (CNN). In this study, we set $H = W = 224$. During preprocessing, nearest neighbor interpolation is used to resize each map, preserving the original pixel magnitudes during both upscaling and downscaling. This ensures accurate recovery of the original map after transformation. Post-training, the output maps are

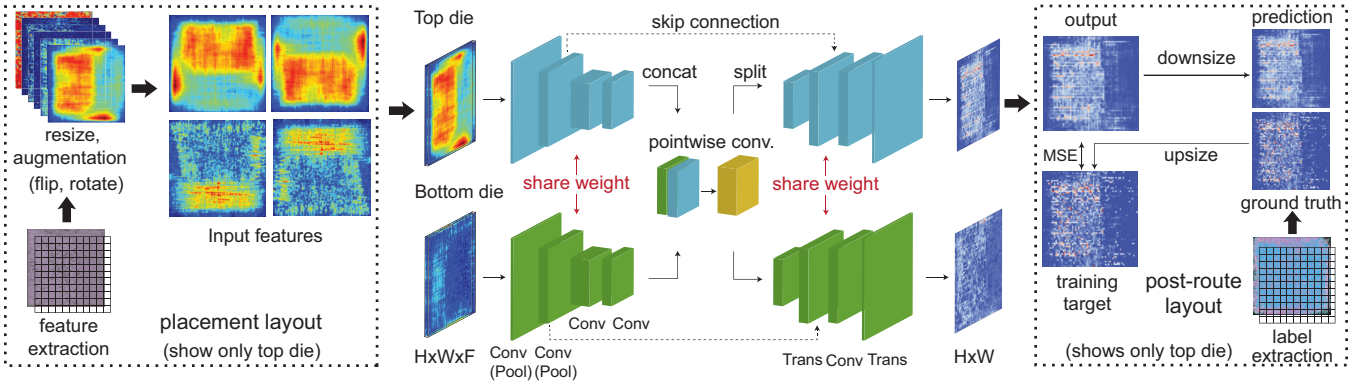


Fig. 3: Our 3D images-to-images Siamese UNet employs a shared encoder-decoder architecture for the top and bottom dies in the F2F 3D IC design. A pointwise communication convolutional layer enables efficient information exchange between the dies, enabling concurrent post-route congestion hotspot prediction in a fully 3D manner.

Algorithm 1 Congestion Prediction

Input: Dataset $\mathcal{D} = \{(F_0^i, F_1^i, C_0^i, C_1^i)\}$ with features $F_0^i, F_1^i \in \mathbb{R}^{h\times w\times 7}$ and targets $C_0^i, C_1^i \in \mathbb{R}^{h\times w}$
Output: Predicted congestion maps $\{\hat{C}_0^i, \hat{C}_1^i \in \mathbb{R}^{h\times w}\}$
Resize features: $F_d^i \in \mathbb{R}^{h\times w\times 7} \rightarrow \mathbb{R}^{H\times W\times 7} \quad \forall d \in \{0,1\}$
Resize targets: $C_d^i \in \mathbb{R}^{h\times w} \rightarrow \mathbb{R}^{H\times W} \quad \forall d \in \{0,1\}$
Initialize SiaUNet with parameters θ
while *not converged* **do**
  $\{\hat{C}_0^i, \hat{C}_1^i\} \leftarrow \text{SiaUNet}(F_0^i, F_1^i; \theta)$
  Compute loss: $L = \frac{1}{2}\sum_{d\in\{0,1\}} \sqrt{\frac{1}{HW}\|\hat{C}_d^i - C_d^i\|_F^2}$
  Update parameters: $\theta \leftarrow \theta - \eta\nabla_\theta L$
end while

resized back to their original dimensions for inference, maintaining consistency with the initial grid structure and ensuring compatibility with subsequent design analyses. To improve the model's robustness to layout orientation, we applied random transformations—-rotations by 0°, 90°, 180°, and 270°, as well as horizontal and vertical flipping—-during training. This data augmentation strategy increases the diversity of training samples by a factor of eight and enhances the model's generalizability to various layout orientations.

C. Overview of Our Prediction Model

Our customized 3D prediction model architecture, illustrated in Figure 3, utilizes feature maps from the 3D global placements of both dies as inputs to predict their post-route congestion maps. The model is built on the UNet architecture, which is well-suited for image-to-image prediction tasks. It features a downsampling encoder and an upsampling decoder: the encoder progressively reduces the spatial dimensions of the input images while increasing the number of feature channels to capture contextual features, and the decoder restores the spatial dimensions by reducing the feature channels through transposed convolutions, ultimately generating the predicted congestion maps. To retain finer details and mitigate information loss, skip connections are incorporated between the encoder and decoder.

Given the interchangeable nature of the top and bottom dies in our face-to-face bonded design, we propose a customized Siamese 3D UNet architecture. This architecture employs shared weights for the encoders and decoders of both dies, ensuring identical feature extraction and reconstruction processes. To capture inter-die dependencies, we introduce a communication layer between the encoder and decoder. This layer merges the encoder outputs from both dies, processes them through pointwise convolution to enable inter-die channel communication, and subsequently splits the processed

Algorithm 2 Differentiable Congestion Optimization (DCO)

Input: Trained congestion predictor SiaUNet*, initial 3D placement $\mathbf{P} = [\mathbf{x}, \mathbf{y}, \mathbf{z}] \in \mathbb{R}^{N\times 3}$, netlist graph $\mathbf{G}$, cell attributes $\mathbf{h}$
Output: Optimized 3D placement $\mathbf{P}^*$
Initialize GNN parameters θ
while *iter* $<$ *max_iter* and not converged **do**
  $\mathbf{P}' \leftarrow \text{GNN}_\theta(\mathbf{G}, \mathbf{h})$ ▷ Updated 3D placement
  $L_{\text{disp.}}, L_{\text{ovlp.}}, L_{\text{cutsize}} \leftarrow$ Compute losses$(\mathbf{P}, \mathbf{P}')$
  $F_d \leftarrow$ Generate features$(\mathbf{P}') \quad \forall d \in \{0,1\}$
  $C'_d \leftarrow \text{SiaUNet}^*(F_d) \quad \forall d \in \{0,1\}$ ▷ Congestion prediction
  $L_{\text{cong.}} \leftarrow$ Compute congestion loss(C'_d)
  $L_{\text{total}} = \alpha L_{\text{disp.}} + \beta L_{\text{ovlp.}} + \gamma L_{\text{cutsize}} + \delta L_{\text{cong.}}$
  $\theta^{i+1} \leftarrow \theta^i - \eta \cdot \frac{\partial L_{\text{total}}}{\partial \theta}$ ▷ Gradient update
end while
return $\mathbf{P}^* \leftarrow \text{GNN}_{\theta^*}(\mathbf{G}, \mathbf{h})$ ▷ Optimized 3D placement

features back into two streams for decoding. Overall, our model is an images-to-images framework that takes the feature map tensors of both dies, $\mathbf{F}_0$, $\mathbf{F}_1 \in \mathbb{R}^{H\times W\times 7}$, as input, and outputs the corresponding post-route congestion maps, $\mathbf{C}_0$, $\mathbf{C}_1 \in \mathbb{R}^{H\times W}$; we set the dimensions to $H = W = 224$.

D. Training Our Prediction Model

The complete training procedure is described in Algorithm 1. The training of our prediction model is guided by the sum of the root mean squared Frobenius losses between the predicted congestion maps, $\hat{\mathbf{C}} \in \mathbb{R}^{H\times W}$, and the ground truth label maps, $\mathbf{C} \in \mathbb{R}^{H\times W}$ for both dies. This objective function is expressed as follows:

$$L = \frac{1}{2}\sum_{d\in\{0,1\}} \sqrt{\frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(\hat{\mathbf{C}}_d(i,j) - \mathbf{C}_d(i,j)\right)^2} \quad (4)$$

IV. Differentiable Congestion Optimization

Our optimization framework is fully differentiable and three-dimensional, allowing for direct, gradient-based refinement of cell placement to enhance congestion metrics and placement quality.

The optimization consists of the following core steps:

1) **GNN-based 3D Cell Spreading:** A GNN predicts updated cell locations to mitigate congestion and enhance placement quality.
2) **Differentiable Loss Functions:** Custom loss functions—including congestion, displacement, overlap, and mincut—guide the GNN during training.
3) **End-to-End Optimization:** Gradients are backpropagated to iteratively update the GNN using gradient descent.

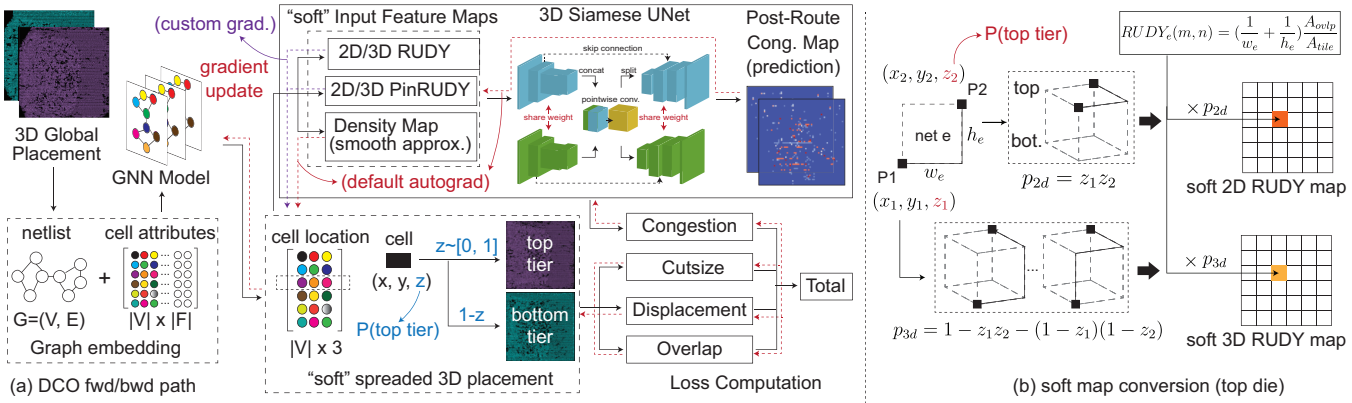


Fig. 4: Differentiable 3D Congestion Optimization Flow: The process starts with a 3D global placement result. The GNN predicts new cell locations as soft probabilistic assignments, partially contributing to both top and bottom tiers. These locations are converted into "soft" feature maps and processed by a 3D Siamese U-Net to predict post-route congestion. The predicted congestion guides GNN optimization with other objectives. A custom backward function handles non-differentiable components, enabling gradient-based optimization of cell locations to achieve design goals.

TABLE II: Initial handcrafted features of each node in the netlist graph.

features	descriptions
wst slack	worst slack of cell
wst output slew	maximum transition of output pin
wst input slew	maximum transition of input pin
drv net power	switching power of driving net
int power	cell internal power
leakage	cell leakage power
width	cell width
height	cell height

The optimization iterates until convergence, as outlined in Algorithm 2. Starting from a 3D global placement, the GNN predicts updated cell locations as soft probabilistic assignments, allowing cells to contribute to both tiers. These locations are transformed into feature maps and passed to the 3D Siamese U-Net to predict post-route congestion. The predicted congestion, along with other objectives, guides GNN optimization. To handle non-differentiable components, a custom backward function enables direct gradient-based refinement of cell locations for multi-objective optimization.

A. Differentiable 3D Cell Spreading

Our cell spreading approach jointly optimizes routability and placement quality using differentiable loss functions that balance congestion, overlap, displacement, and cutsize. Their differentiability enables gradient-based optimization of cell locations. Unlike 2D methods limited to horizontal and vertical spreading, our 3D solution redistributes cells across dies. This added flexibility in the z-direction effectively resolves congestion hotspots that are unsolvable in traditional 2D layouts. To efficiently optimize cell placement, we avoid learning independent (x, y, z) coordinates per cell, which would scale poorly for large netlists with millions of parameters. Instead, we utilize a GNN consisting of three Graph Convolutional Network (GCN) layers with shared weights across all cells. This approach leverages cell connectivity and intrinsic characteristics (Table II), allowing globally informed updates and stable convergence. The GNN predicts (x, y, z), where x and y are continuous 2D positions, and $z \in [0, 1]$ represents the probability of assigning a cell to the top die (z) or bottom die $(1 - z)$. This probabilistic z promotes differentiable optimization across tiers, allowing each cell to influence both dies before final hard assignment via $z = \mathbb{1}z \geq 0.5$. During training, z guides soft tier assignments in feature and density maps. For RUDY maps (Fig. 4b), the 2D contribution is weighted by $\prod p \in e z_p$ (top) or $\prod_{p \in e}(1 - z_p)$ (bottom), while the 3D contribution is weighted by $1 - \prod_{p \in e} z_p - \prod_{p \in e}(1 - z_p)$. These maps inform loss computations for gradient-based optimization.

B. Congestion Loss

After the GNN generates new cell locations, the input feature maps (Sections III-B1 and IV-A) are processed by the model (Figure 3) to predict congestion maps $\mathbf{C} \in \mathbb{R}^{H \times W \times 2}$ for the top and bottom dies. The congestion penalty is calculated using Eq. 4. The gradient with respect to the GNN parameters θ is computed via the chain rule:

$$\frac{\partial L}{\partial \theta} = \sum_{d \in [\text{top,bottom}]} \frac{\partial L}{\partial \mathbf{C}_d} \cdot \frac{\partial \mathbf{C}_d}{\partial \mathbf{F}_d} \cdot \frac{\partial \mathbf{F}_d}{\partial \mathbf{X}_{\text{cells}}} \cdot \frac{\partial \mathbf{X}_{\text{cells}}}{\partial \theta} \tag{5}$$

$\partial L / \partial \mathbf{C}_d$ measures how the loss changes with respect to the predicted congestion map $\mathbf{C}_d$. The term $\partial \mathbf{C}_d / \partial \mathbf{F}_d$ represents the sensitivity of congestion predictions to changes in the feature maps $\mathbf{F}_d$, while $\partial \mathbf{F}_d / \partial \mathbf{X}_{\text{cells}}$ describes how feature maps are influenced by cell positions. Since $\partial \mathbf{F}_d / \partial \mathbf{X}_{\text{cells}}$ is non-differentiable at grid boundaries, we implement a custom PyTorch backward function that approximates gradients using subgradients for non-differentiable components while leveraging PyTorch's native autograd for smooth terms. For each tile (m, n), we compute the gradient of RUDY values with respect to cell positions by summing contributions from all crossing nets. The sensitivity of a net e's RUDY value in tile (m, n) to the x-position of cell i $\partial RUDY_e(m, n) / \partial x_i$ is given by:

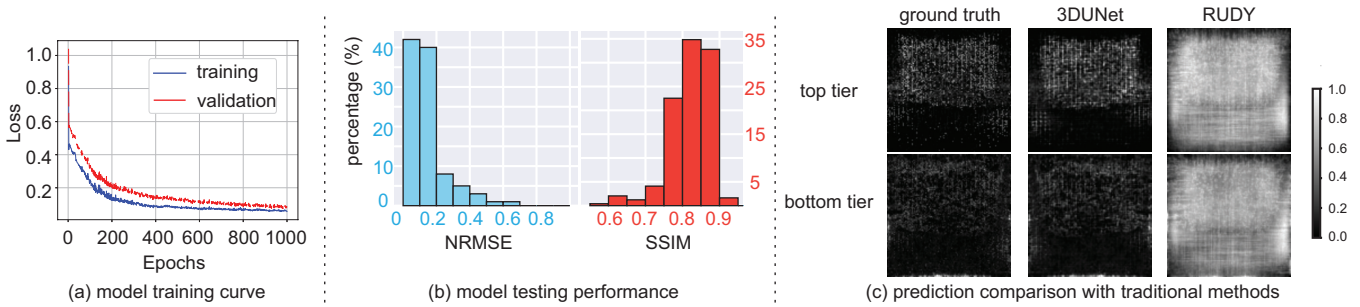
$$\mu_e(m, n) \left[\frac{(w_e - w')h'}{w_e^2} + \frac{h'}{h_e} \right] \frac{1}{A_{\text{GCell}}} (\delta_{ih} - \delta_{il}) \tag{6}$$

where w', h' are the dimensions of the net-tile overlap, A_{GCell} is the tile area, and $\mu_e(m, n)$ indicates whether net e crosses tile (m, n). The term $(\delta_{ih} - \delta_{il})$ represents changes in the net's bounding box due to cell i's movement. Here, δ is the Kronecker delta function, which is nonzero only when cell i contains the leftmost $(\delta_{il} = 1)$ or the rightmost $(\delta_{ih} = 1)$ pin of the net.

C. Cutsize Loss

While 3D spreading allows cell movements between dies to alleviate congestion hotspots, it is essential to minimize the cut size—the number of inter-die connections—as excessive cuts can increase fabrication costs and degrade circuit performance. To address this, we incorporate a min-cut loss into our optimization:

$$L_{cut}(T, B) = \frac{\text{cut}(T, B)}{\deg(T)} + \frac{\text{cut}(T, B)}{\deg(B)} \tag{7}$$



(a) model training curve (b) model testing performance (c) prediction comparison with traditional methods

Fig. 5: Evaluation of prediction results: (a) Training and testing loss curves, (b) X-axis denotes absolute NRMSE/SSIM values; y-axis denotes percentage of testing samples. Both metrics fall in desired regions. (c) Comparison of predicted, traditional, and ground-truth congestion maps.

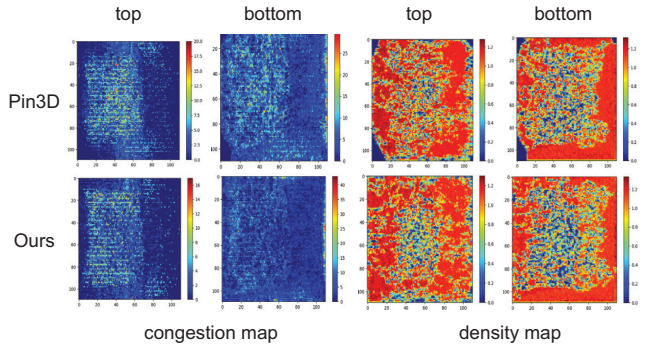


Fig. 6: Post-route congestion and density map of Pin3D vs. ours

Here, $\text{cut}(T, B)$ counts the edges connecting cells between the top die T and the bottom die B, while $\deg(T)$ and $\deg(B)$ represent the total connections within each die. This normalized formulation balances inter-die and intra-die connectivity, ensuring a manageable number of inter-die connections for fabrication.

D. Overlap Loss

To address overlap in cell spreading efficiently, we use a density loss metric to promote spatial separation. Since density function $D_b(x, y)$ are typically non-smooth and non-differentiable, we apply smoothing using bell-shaped potential functions for both the x and y directions. These functions are piecewise-defined based on the center-to-center distance d_x or d_y between the block and the bin:

$$p_x(b,v) = \begin{cases} b\left(1 - ad_x^2\right) & \text{if } 0 \le d_x \le w_b + \frac{w_v}{2}, \\ b\left(d_x - \frac{w_v}{2} - 2w_b\right)^2 & \text{if } 2w_b + \frac{w_v}{2} \le d_x \le \frac{w_b + 2w_v}{2}, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where w_b and w_v are the block and bin width, and a and b are smoothing parameter:

$$a = \frac{4}{(w_v + 2w_b)(w_v + 4w_b)}, \quad b = \frac{2}{w_b(w_v + 4w_b)}. \tag{9}$$

Our objective is the density function $\hat{D}_b(x, y)$:

$$\hat{D}_b(x,y) = \sum_{v \in V} c_v p_x(b,v) p_y(b,v). \tag{10}$$

E. Displacement Loss

To preserve the quality of the optimized 2D positions in initial 3D global placement, we constructed a displacement loss function to constrain cell movement. Displacement loss minimizes the deviation of cells' current positions from their original locations, preserving the quality of the 3D global placement. The loss is defined as:

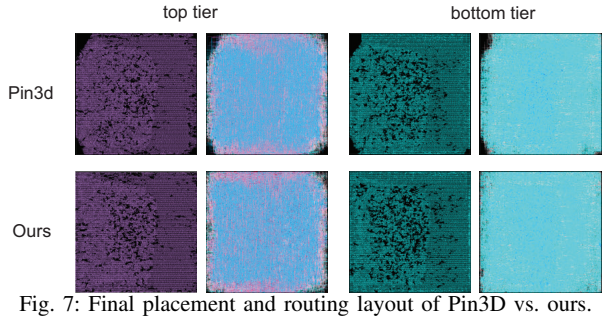


Fig. 7: Final placement and routing layout of Pin3D vs. ours.

$$L_{disp.} = \sum_i (x_i - x_i^o)^2 + (y_i - y_i^o)^2 \tag{11}$$

V. Experimental Results

We perform thorough experiments by comparing our DCO-3D flow against Pin-3D [11], the state-of-the-art 3D design flow and its improved variations on 6 industrial designs, including RocketCore, LDPC, AES, ECG, DMA, and VGA. All the benchmarks are synthesized under a commercial 3nm technology node using *Synopsys Design Compiler* with 1μm pitch face-to-face 3D hybrid bonding. We leverage *Synopsys ICC2* to perform pseudo-2D and 3D P&R. The Siamese network and the differentiable GNN-based cell spreader, are implemented in Python using PyTorch and PyTorch Geometric.

A. Prediction Results

To train the proposed Siamese architecture, illustrated in Figure 3, we construct our dataset as described in Section III-A, reserving 20% as the testing set. To evaluate the spatial accuracy and visual similarity of the model's map predictions, we use two metrics for 2D grid signals: normalized root mean square error (NRMSE) and structural similarity index (SSIM). NRMSE measures the discrepancy between predictions and observed values, with lower values indicating higher accuracy. For image prediction tasks, an NRMSE below 0.2 signifies close alignment with actual images. SSIM quantifies structural similarity between images, ranging from -1 to 1, where 1 denotes identical images. SSIM values above 0.7 are sufficient.

Figure 5 presents our Siamese UNet-based model's prediction results, including training curves, metrics, and visualizations. As shown in Figure 5 (b), over 85% of test samples achieve NRMSE below 0.2 and SSIM above 0.8, indicating high prediction accuracy. Figure 5 (c) compares our model with the RUDY estimator on a test sample (AES design), with pixel values normalized to $[0, 1]$ for fairness. While RUDY poorly correlates with the ground truth in 3D ICs, our 3D IC-customized Siamese UNet shows far higher similarity, validating its effectiveness as a reliable proxy for post-route routability optimization during 3D global placement.

TABLE III: Detailed comparisons of optimization results between state-of-the-art 3D flow Pin-3D [11], Pin-3D with *Synopsys ICC2* congestion-focus optimization (Pin-3D + Cong.), Pin-3D with Bayesian Optimization (Pin-3D + BO), and the proposed DCO-3D flow over 6 industrial benchmarks in a commercial $3nm$ technology node. The best metric in each column is colored in red. Note that we use the exact same *Synopsys ICC2* seed across all experiments to completely remove run-to-run non-determinism.

evaluation stage	after 3D placement optimization				after signoff optimization (end-of-flow)			
	overflow	ovf. gcell%	H ovf.	V ovf.	setup wns (ps)	setup tns (ps)	total power (mW)	WL (μm)
DMA (#cells: 13K, #nets: 14K, #IO: 961)								
Pin3D [11]	3389	26.17	1914	1475	-24.45	-5277	11.3	25572.14
Pin3D + Cong.	2352	23.62	693	1659	-42.67	-6439	11.2	26705.18
Pin3D + BO	2214	21.96	573	1641	-58.87	-17503	11.2	26387.22
DCO-3D (ours)	2018 (-40.45%)	20.15	586	1432	-22.05 (-9.82%)	-4443 (-15.79%)	10.9 (-3.54%)	25076.70
AES (#cells: 114K, #nets: 114K, #IO: 390)								
Pin3D [11]	34158	32.11	18149	16009	-35.64	-10877	75.1	243430.25
Pin3D + Cong.	21018	23.13	5617	15401	-46.27	-11714	74.4	246183.06
Pin3D + BO	19362	22.07	3872	15490	-52.28	-12049	74.1	252131.34
DCO-3D (ours)	18026 (-47.23%)	21.99	2498	15528	-27.15 (-23.82%)	-7928 (-27.11%)	72.6 (-3.33%)	233401.63
ECG (#cells: 83K, #nets: 84K, #IO: 1.7K)								
Pin3D [11]	15083	19.32	548	14535	-54.96	-41163	78.0	200362.67
Pin3D + Cong.	14137	18.49	559	13578	-75.25	-90069	77.9	199021.49
Pin3D + BO	13929	18.27	607	13322	-78.28	-47713	75.9	207186.17
DCO-3D (ours)	13686 (-9.26%)	17.06	1520	12166	-41.47 (-24.55%)	-12394 (-69.89%)	75.9 (-2.69%)	198998.32
LDPC (#cells: 39K, #nets: 41K, #IO: 4.1K)								
Pin3D [11]	17666	38.90	3318	13709	-154	-145389	71.3	211438.43
Pin3D + Cong.	13637	33.81	1335	12302	-160.32	-167845	71.1	214667.37
Pin3D + BO	13091	31.43	1668	11423	-169.13	-193429	71.0	220390.83
DCO-3D (ours)	12511 (-26.52%)	29.77	2685	9826	-75.68 (-50.86%)	-69314 (-52.33%)	67.6 (-5.19%)	209304.86
VGA (#cells: 52K, #nets: 52K, #IO: 184)								
Pin3D [11]	38523	34.38	2766	35757	-20.16	-114.77	17.1	124202.71
Pin3D + Cong.	34005	38.69	929	33076	-150.53	-4505	16.9	138850.31
Pin3D + BO	33969	38.99	700	33269	-227.41	-12170	16.7	147160.69
DCO-3D (ours)	32578 (-15.43%)	37.42	905	31673	-10.3 (-48.91%)	-25.14 (-78.10%)	16.5 (-3.51%)	123712.41
Rocket (#cells: 120K, #nets: 120K, #IO: 379)								
Pin3D [11]	27742	7.00	818	26924	-308.95	-20321	31.2	345382.06
Pin3D + Cong.	22958	5.81	627	22331	-605.58	-59599	30.6	350021.31
Pin3D + BO	22286	5.73	613	21673	-779.8	-183225	31.2	363796.28
DCO-3D (ours)	17963 (-35.25%)	4.49	1460	16503	-147.33 (-52.31%)	-2800 (-86.22%)	29.7 (-4.81%)	328475.40

B. Optimization Results

Figure 6 and 7 show the post-route layout, congestion maps, and density maps for both dies of the optimized and unoptimized designs using the LDPC benchmark. In the optimized design, cells are more effectively distributed, leading to improved congestion and density without significantly increasing wirelength (Table III), due to reduced detours. This translates into substantial full-chip PPA improvements at the end of the flow, as detailed in Table III.

To address limited congestion awareness in the standard Pin-3D flow, we introduce two enhanced benchmarks for comprehensive comparisons. The first enhancement, "Pin-3D + Cong.", enables *ICC2* congestion-driven placement at the highest effort during the 3D placement stage. The second benchmark, "Pin-3D + BO," augments Pin-3D with Bayesian Optimization (BO) techniques from [19] to optimize tool parameters listed in Table I.

The detailed optimization results of Pin-3D, its variants, and our DCO-3D are presented in Table III. While DCO-3D demonstrates immediate congestion optimization benefits following the 3D placement stage, the key achievement lies in translating these benefits into superior 3D PPA outcomes at the end of the 3D signoff step. Across six industrial benchmarks using a foundry $3nm$ technology node, DCO-3D consistently delivers the best end-of-flow PPA metrics, with setup TNS improved by up to 86% and total power reduced by up to 4.8%. These results highlight how our early congestion optimization leads to significant full-chip PPA improvements in 3D ICs.

C. Why Does DCO-3D Work While Others Do Not?

Unlike "Pin-3D + Cong." and "Pin-3D + BO," which rely on ad-hoc congestion optimization, DCO-3D introduces differentiable congestion objectives optimized directly via gradient-based methods. By enabling cross-die cell spreading, DCO-3D redistributes congestion and substantially reduces routing overflow. Furthermore, by co-optimizing congestion with other objectives, DCO-3D minimizes congestion without compromising overall design quality. As shown in Table III, DCO-3D achieves notable PPA improvements, outperforming BO and commercial auto-fixing features, which often compromise critical PPA metrics.

VI. Conclusion

In this paper, we introduce DCO-3D, the first 3D flow that employs ML techniques to tackle routability issues in advanced technology nodes. DCO-3D accurately forecasts routability during the early stages of 3D global placement and a implements differentiable 3D cell spreading treatment. Extensive experiments show that DCO-3D not only effectively reduces congestion but also significantly improves PPA metrics during the final signoff evaluation.

VII. Acknowledgement

This work was supported by the Semiconductor Research Corporation under the JUMP 2.0 Center Program (CHIMES), Samsung Advanced Institute of Technology (SAIT) under the AI for Semiconductors Program, and the Ministry of Trade, Industry & Energy of South Korea (1415187652, RS-2023-00234159).

REFERENCES

[1] H. Park, B. W. Ku, K. Chang, D. E. Shim, and S. K. Lim, "Pseudo-3d approaches for commercial-grade rtl-to-gds tool flow targeting monolithic 3d ics," in *Proceedings of the 2020 International Symposium on Physical Design*, pp. 47–54, 202.

[2] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena, "Routability optimization for industrial designs at sub-14nm process nodes using machine learning," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, pp. 15–21, 2017.

[3] W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath, and K. Samadi, "BEOL stack-aware routability prediction from placement using data mining techniques," *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pp. 41–48, 2016.

[4] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "RouteNet: Routability prediction for Mixed-Size Designs Using Convolutional Neural Network," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2018.

[5] S. Kim, H. Park, K. Baek, K. Choi, and T. Kim, "Methodology of Resolving Design Rule Checking Violations Coupled with Fully Compatible Prediction Model," in *Proceedings of the 2024 International Symposium on Physical Design*, ISPD '24, (New York, NY, USA), p. 103–111, Association for Computing Machinery, 2024.

[6] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu, "DRC Hotspot Prediction at Sub-10nm Process Nodes Using Customized Convolutional Network," in *Proceedings of the 2020 International Symposium on Physical Design*, ISPD '20, (New York, NY, USA), p. 135–142, Association for Computing Machinery, 2020.

[7] C. Yu and Z. Zhang, "Painting on Placement: Forecasting Routing Congestion using Conditional Generative Adversarial Nets," in *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[8] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. A. Iyer, and D. Z. Pan, "High-Definition Routing Congestion Prediction for Large-Scale FPGAs," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 26–31, 2020.

[9] Y.-H. Huang, Z. Xie, G.-Q. Fang, T.-C. Yu, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routability-Driven Macro Placement with Embedded CNN-Based Prediction Model," pp. 180–185, 03 2019.

[10] G. Koch, R. Zemel, R. Salakhutdinov, *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2, Lille, 2015.

[11] S. S. Kiran Pentapati *et al.*, "Pin-3D: A Physical Synthesis and Post-Layout Optimization Flow for Heterogeneous Monolithic 3D ICs," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2020.

[12] J. Cong and G. Luo, "A multilevel analytical placement for 3D ICs," in *2009 Asia and South Pacific Design Automation Conference*, pp. 361–366, 2009.

[13] Y.-J. Chen, Y.-S. Chen, W.-C. Tseng, C.-Y. Chiang, Y.-H. Lo, and Y.-W. Chang, "Late Breaking Results: Analytical Placement for 3D ICs with Multiple Manufacturing Technologies," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–2, 2023.

[14] J. Lu, H. Zhuang, I. Kang, P. Chen, and C.-K. Cheng, "ePlace-3D: Electrostatics based Placement for 3D-ICs," in *Proceedings of the 2016 on International Symposium on Physical Design*, ISPD '16, (New York, NY, USA), p. 11–18, Association for Computing Machinery, 2016.

[15] M.-K. Hsu, V. Balabanov, and Y.-W. Chang, "TSV-Aware Analytical Placement for 3-D IC Designs Based on a Novel Weighted-Average Wirelength Model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 4, pp. 497–509, 2013.

[16] X. Zhao, S. Chen, Y. Qiu, J. Li, Z. Huang, B. Xie, X. Li, and Y. Bao, "iPL-3D: A Novel Bilevel Programming Model for Die-to-Die Placement," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023.

[17] H. Park *et al.*, "Pseudo-3D Physical Design Flow for Monolithic 3D ICs: Comparisons and Enhancements," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, jun 2021.

[18] P. Spindler and F. M. Johannes, "Fast and Accurate Routing Demand Estimation for Efficient Routability-driven Placement," in *2007 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1–6, 2007.

[19] Y. Ma, Z. Yu, and B. Yu, "CAD Tool Design Space Exploration via Bayesian Optimization," in *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6, 2019.

Contention-Aware Forecasting of Energy Efficiency through Sequence-Based Models in Modern Heterogeneous Processors

Mohammed Bakr Sikal*, Jeferson González-Gómez*†, Heba Khdr*, Jörg Henkel*
* *Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*
† *Instituto Tecnológico de Costa Rica (TEC), Cartago, Costa Rica*
{bakr.sikal, jeferson.gonzalez, heba.khdr, henkel}@kit.edu

***Abstract*—We present *EffiCast*, the first methodology for contention-aware energy efficiency forecasting in clustered heterogeneous processors using sequence-based models. Through extensive experimental analysis of energy efficiency sensitivities across core types, voltage/frequency (V/f) levels, application phases, and resource contention scenarios, *EffiCast* uncovers key factors driving energy efficiency variability in modern heterogeneous processors. Leveraging structured data generation and advanced LSTM- and Transformer-based models, *EffiCast* achieves unprecedented accuracy while outperforming state-of-the-art predictive techniques. Deployed on a real heterogenous processor with Intel's oneDNN acceleration, *EffiCast* delivers inference latencies as low as 1.82 ms per sequence, enabling seamless integration into proactive resource management frameworks. With the ability to forecast future system states under dynamic workloads, *EffiCast* sets a new standard for energy efficiency optimization in energy-constrained application domains.**

I. Introduction

The demand for energy-efficient computing has never been more critical in modern systems [1], i.e., smart industries, self-driving vehicles, etc., driven by the need to operate within stringent energy budgets while maintaining performance. Concurrently, workloads are also becoming increasingly diverse and complex, exhibiting varying characteristics in terms of computational intensity memory requirements. This diversity complicates the task of achieving energy efficiency, as different applications demand resource management approaches that are tailored to their characteristics.

Heterogeneous processors, such as Intel's *Alder Lake* Performance Hybrid Architecture [2], have emerged as a promising architectural solution to this challenge, combining high-**P**erformance cores with energy-**E**fficient ones. To exploit the heterogeneity of such processors, a system-level resource management technique should adjust the mapping of applications to the most suitable core type, depending on their characteristics. However, as will be shown in the next motivational example, the decision-making process for selecting the core type that offers the highest energy efficiency for each application *is not trivial*, especially when combined with V/f level adjustment.

Figure 1 illustrates the execution time and system-wide energy efficiency of two applications from the PARSEC [3] and SPLASH-2 [4] benchmark suites, executed across different V/f levels and core types. The results are normalized against the slowest execution time and the most energy-efficient execution of the same application. All experiments are conducted on a real heterogeneous processor[1]. The energy efficiency is quantified as instructions per Joule (IPJ), i.e, the total number of instructions executed per unit of system-wide energy consumption in Joules, where a higher IPJ indicates a more energy-efficient execution. The application *fft* demonstrates a clear sensitivity to both core type and V/f levels, achieving its *fastest* and *most energy-efficient* execution on a P-core. In contrast, the application *dedup* exhibits a clearly different behavior; several executions on E-cores achieve comparable performance and energy efficiency to P-cores. Notably, in the mid-range V/f levels, the E-core execution of *dedup* outperform the P-cores in energy efficiency.

[1]A Linux-based heterogeneous Intel® Core™ i9 12th generation Alder Lake processor. Further setup details are explained in Section II.

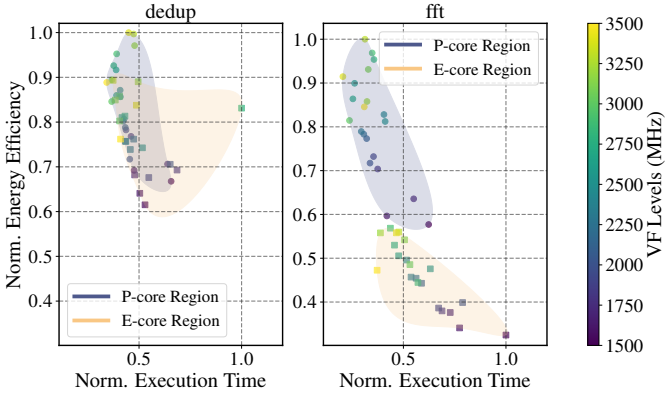


Fig. 1. Executing applications on a heterogeneous processor[1] shows different performance and energy efficiency sensitivities to changes in core types and V/f levels. The same color coding refers to the same VF level across both core types and the different colors represent different VF levels. While *fft* exhibits clear performance and energy efficiency sensitivities to the core types and V/f level changes with a higher energy efficiency on the P-cores, *dedup* shows notably different trends.

This example shows that simple heuristic-based mapping and dynamic voltage and frequency scaling (DVFS) strategies cannot fully leverage the capabilities of modern processors, as it would *fail* to adapt effectively to the nuanced performance and energy characteristics of diverse applications, highlighting the need for more advanced resource management approaches. The state of the art increasingly recognizes this fact. Machine learning (ML)-based and predictive resource management techniques are becoming essential to help proactively predicting the *impact* of core type or/and V/f level selection. For example, prior works have explored code instrumentation [5], [6] as a characterization means of applications. This enabled the imitation learning-based runtime optimization technique in [7] to consider the impact of heterogeneity of core types on energy. However, these methods rely on prior knowledge of applications and require access to their source code, an assumption that is *impractical* in most modern computing environments. Other approaches have employed supervised learning models to predict the performance and power sensitivities of applications [8] to V/f level changes [9], albeit evaluated only in homogeneous processor environments. As will be discussed in Sections I-A and V, predicting energy efficiency sensitivity to *both* the core type and V/f level remains an open challenge in the literature.

A. Challenges of Energy Efficiency Prediction

Predicting energy efficiency in heterogeneous processors is a highly complex problem, driven by the following intertwined challenges. **First**, as shown in Fig. 1, energy and performance sensitivities vary significantly depending on the characteristics of applications and the underlying hardware. Even when known and profiled, it is not trivial to select the optimal core type and V/f level for some applications, e.g., *dedup*. At runtime, however, applications may be unknown, further complicating the task. Without prior knowledge of application

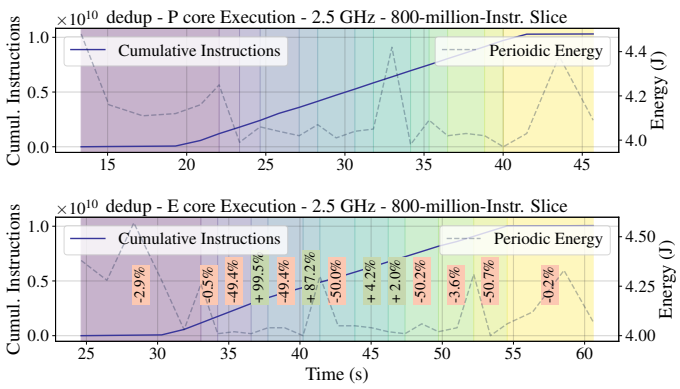


Fig. 2. A zoomed-in view of the phase-level (800 million retired instructions per phase) periodic energy consumption over time for *dedup* shows phase-level energy efficiency improvements (shown in the colored labels) on the E-core compared to the P-core, contrary to the initial observation in Fig. 1, where the overall execution of *dedup* was consistently more efficient on the P-cores.

behavior, accurately predicting energy efficiency becomes a formidable challenge, requiring robust models that are able to generalize across unseen workloads and scenarios that occur at runtime.

Secondly, applications are not monolithic in their behavior; instead, they transition through distinct phases during execution, each exhibiting different resource utilization patterns and sensitivities to system heterogeneity. As we will demonstrate in the analysis section Section II, the same application can display completely different energy efficiency characteristics depending on its execution phase, requiring models to account for this variability.

Third, and even more challenging, is the rapidity at which applications can transition between phases. When predicting the impact of a decision—such as adjusting the V/f level or migrating an application to a different core—the system state may have already evolved by the time the decision is applied. Traditional machine learning prediction techniques that rely on static or *short-sighted* input data fail to capture this temporal dynamic. Instead, as we will demonstrate in Section III, forecasting future system states is crucial to making proactive and effective decisions at runtime.

Finally, heterogeneous processors feature complex *shared* resource hierarchies, including caches and memory bandwidth, which introduce contention both within and across clusters. This contention is highly dynamic, as it depends not only on the co-running applications on the same cluster, but also on the phase transitions of applications running on other clusters. Accurately capturing these contention effects and their impact on energy efficiency adds another layer of complexity to the prediction problem.

To address these challenges, we start by performing an extensive analysis to understand the different factors impacting the predictability of energy efficiency sensitivity of applications to the heterogeneity of modern processors. Guided by the insights from our analysis, we present for the first time *EffiCast*, a novel methodology for contention-aware energy efficiency and sequence-based forecasting for heterogeneous processors. The key contributions of this work are as follows:

- We perform a comprehensive analysis of energy efficiency sensitivities across core types, V/f levels, application phases, and resource contention scenarios, uncovering key factors that impact energy efficiency prediction.
- We propose EffiCast, the first methodology for training energy efficiency forecasting models, incorporating structured data generation and sequence-based techniques to predict future system states under dynamic and unseen scenarios.
- We build, train, and evaluate LSTM- and Transformer-based forecasting models on a real heterogeneous processor platform, demonstrating superior accuracy over traditional state-of-the-art predictive models.

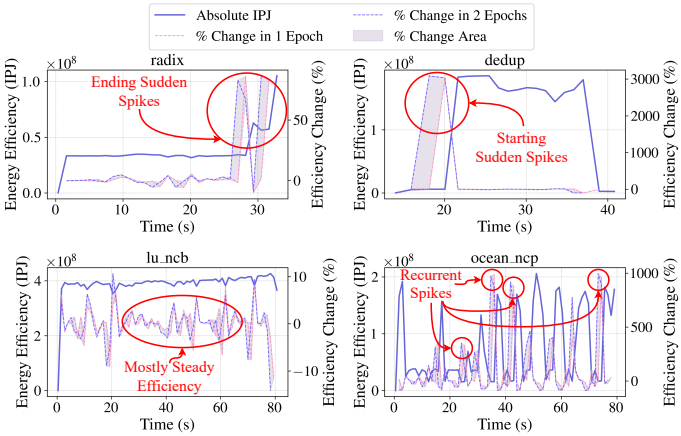


Fig. 3. Absolute energy efficiency (IPJ) and percentage changes over one and two epochs for four representative applications. Shaded areas highlight variability in energy efficiency, reflecting dynamic behaviors such as starting spikes (*dedup*), recurrent spikes (*ocean_ncp*), and abrupt phase transitions (*radix*). These diverse patterns underscore the need for sequence-based models capable of capturing temporal dependencies and dynamic phase transitions.

II. Analyzing Energy Efficiency Sensitivity to Heterogeneity

To expand on the observations and challenges discussed in Section I, we conduct in the following section a comprehensive analysis of energy efficiency sensitivities across core types, V/f levels, application phases, and resource contention scenarios. Experiments in this section run PARSEC [3] and SPLASH-2 [4] benchmark suites on a Linux-based heterogeneous Intel® Core™ i9 12th generation Alder Lake processor, with Performance Monitoring Counters (PMCs) collected every 100 ms using *perf* [10] and system-wide energy consumption monitored via the Running Average Power Limit (RAPL) interface at the same interval.

A. Volatility of Phase-Level Sensitivities

First, we analyze the phase-level sensitivity of *dedup* to core types and V/f levels, as to compare it with the observations from Fig. 1, when it was executed on a single core type throughout its execution. Fig. 2 illustrates the cumulative retired instructions and periodic energy consumption for *dedup* at 2.5 GHz when executed on both a P-core and an E-core. To enable a phase-level analysis, the retired instructions are divided into slices of 800 million instructions each. As expected, the execution times of different phases vary significantly due to the fluctuating instructions per second (IPS). Since both cores execute the same total number of instructions, slices are matched across both P-core and E-core executions, with color-coding to enable an easy comparison. The highlighted labels show the per-phase *energy efficiency improvements* of the E-core execution compared to the P-core execution for the same phase. While Fig. 1 showed the P-core as the energy-efficient option in general, i.e., considering that the application executes from start to finish on the P-core, Fig. 2 reveals that some phases on the E-core are **up to 2x more energy-efficient** than on the P-core.

At runtime, before changing the application-to-core mapping or the V/f level of a core, a resource management technique should be able to *evaluate* whether such a decision would indeed improve the energy efficiency of the system at that specific phase. To enable this *proactiveness*, there is a need to predict the *phase-level sensitivities of applications*.

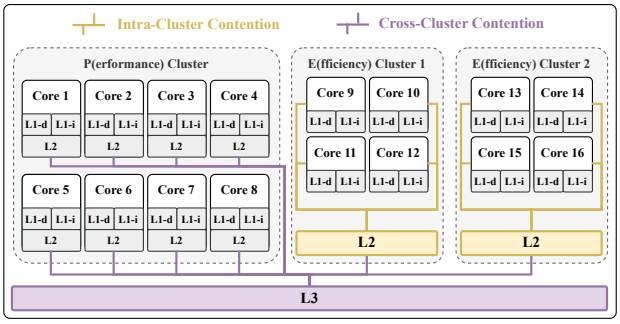


Fig. 4. Illustration of the heterogeneous Intel® Core™ i9 12th generation Alder Lake processor, highlights the complexity of modern memory hierarchies. The shared L2 cache within each E cluster and the shared L3 cache between all clusters create both intra-cluster and cross-cluster contention, respectively.

B. Temporal Variability in Energy Efficiency

The temporal dynamics of energy consumption and performance across different applications lead to intricate patterns and variability in energy efficiency, thereby complicating its prediction at runtime. Figure 3 illustrates the absolute energy efficiency, measured in IPJ, and the percentage changes in energy efficiency over one and two epochs for four representative applications: *radix*, *dedup*, *lu_ncb*, and *ocean_ncp*. The figure was generated by processing periodic data collected during execution, with energy efficiency calculated as the ratio of instructions executed by the application to the total energy consumed by the system.

Figure 3 captures three critical aspects of the temporal variability of energy efficiency over very short time horizons. The dashed lines represent the percentage relative changes in energy efficiency over the next one and two epochs, and provide additional insight into the magnitude and frequency of efficiency variability. The shaded region between the minimum and maximum percentage changes quantifies the potential variability in efficiency. The figure reveals distinct patterns that emerge across the applications. *lu_ncb* exhibits a steady efficiency profile, with very minor fluctuations, maintaining efficiency changes below 10% across its execution. For *radix* and *dedup*, their absolute IPJs remain relatively stable for most of their execution, but exhibit sharp unexpected spikes toward the beginning and end of execution, respectively. The spike in *dedup*'s energy efficiency exceeds a tremendous 3000%. The behavior of *ocean_ncp* is particularly noteworthy, as it displays recurring spikes in energy efficiency, indicative of periodic execution phases with alternating characteristics, with relative IPJ changes of more than 1000%.

C. Intra-Cluster and Cross-Cluster Contention

The complexity of accurately predicting energy efficiency in heterogeneous multi-core processors is further amplified due to dynamic contention at both intra-cluster and cross-cluster levels. Contention arises as co-running applications compete for shared resources, such as caches and memory bandwidth, leading to variable performance and energy behavior.

Figure 4 highlights the shared resource hierarchy of our target processor, illustrating sources of contention. Within clusters, applications compete for resources like L2 caches in E clusters, creating *intra-cluster* contention. Meanwhile, *cross-cluster* contention occurs at shared L3 caches accessed by all cores. This multi-level contention complicates energy sensitivity predictions, as application behavior is influenced by both its own resource usage and that of co-running applications. To explore these effects, we conducted extensive experiments with workloads of up to 16 parallel applications. Figure 5 shows execution times for four applications (*canneal*, *bodytrack*, *blackscholes*, and *water_nsquared*) under varying numbers of background applications.

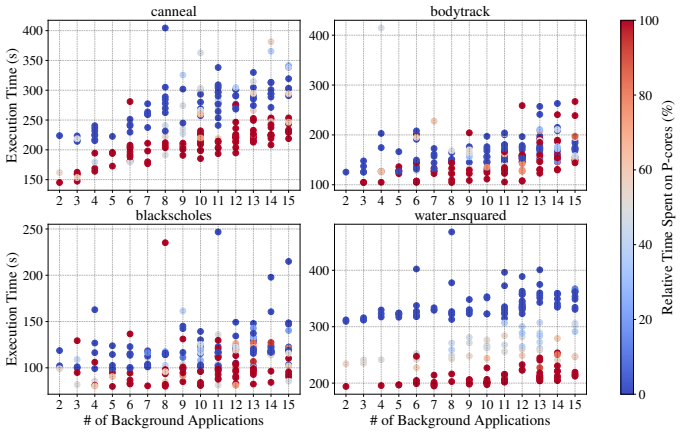


Fig. 5. Execution time of four applications, running with varying numbers of background applications. Random core mapping and periodic migrations reveal the impact of inter- and cross-cluster contention, showing varying but maintained sensitivity across core types with some applications (*canneal*, *water_nsquared*), and significant variability for others (*bodytrack*, *blackscholes*), emphasizing the complexity of contention in heterogeneous processors.

In all experiments, applications were initially mapped randomly to cores, and the application of interest was periodically migrated across different cores types/clusters. Data points are color-coded based on the time spent on P-cores, which typically offer higher performance.

The results reveal several key insights. Execution time does not scale directly with the number of co-running applications, as observed in *canneal* and *water_nsquared*, highlighting that performance degradation depends on the characteristics of co-running applications rather than their number. For *water_nsquared* and *canneal*, P-cores consistently deliver the best performance regardless of contention levels. In contrast, applications like *bodytrack* and *blackscholes* show significant variability between P-core and E-core executions, with performance heavily influenced by both contention levels and migration decisions.

In summary, this analysis demonstrates that accurately modeling energy efficiency sensitivity to modern heterogeneous processors *requires* powerful *forecasting* models capable of capturing *temporal* dynamics of *phase-level sensitivities* under *inter-cluster* and *cross-cluster* contention.

III. EffiCast: Novel Forecasting Methodology

Motivated by our findings and observations in Section II, we present for the first time *EffiCast*, a novel methodology for contention-aware energy efficiency sequence-based forecasting for heterogeneous processors. Our problem formulation and training methodology address all the challenges discussed in Section I-A and leverages the observations from the extensive experimental analysis in Section II. In the following subsections, we present our machine learning problem formulation, training data generation approach, feature engineering methodology, and model architectures.

A. Problem Formulation

The forecasting task aims to predict the energy efficiency of a heterogeneous, clustered multi-core processor system under dynamic workload conditions. Formally, let T represent the number of epochs observed, and $\mathcal{X}_t = \{\mathbf{X}_t^i\}$ represent the set of input features at epoch t, where $\mathbf{X}_t^i \in \mathbb{R}^d$ corresponds to individual features for the i-th application and system-level metrics. The system energy efficiency at epoch t is denoted as E_t, expressed in IPJ. Given the input feature sequence $\{\mathcal{X}_t, \mathcal{X}_{t+1}, \ldots, \mathcal{X}_{t+T}\}$, the task is to predict the energy efficiency E_{t+T+1} of the system *under a hypothetical configuration change*, $\mathbf{C}_{t+T+1}$, where:

- $\mathbf{C}_{t+T+1}$ includes the new mapping of the application of interest and the adjusted V/f level.

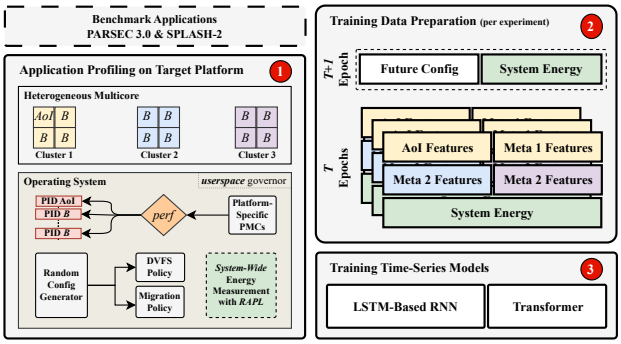


Fig. 6. An oveview of EffiCast, our novel methodology for contention-aware energy efficiency sequence-based forecasting for heterogeneous processors.

- $\mathcal{X}_t$ includes features derived from the application of interest, aggregated cluster metrics, and system-wide metrics.

The objective is to minimize the forecasting error $\mathcal{L}$, which quantifies the mean squared difference between the true energy efficiency E_{t+T+1} and the predicted value $\hat{E}_{t+T+1}$ across all N samples.

B. Training Data Generation

To train the forecasting models, we conduct experiments using applications from the PARSEC and SPLASH-2 benchmark suites. Each experiment randomly samples a set of applications and executes them on a Linux-based heterogeneous Intel® Core™ i9 12th generation Alder Lake processor. PMCs are collected every 100 ms using *perf* [10], while system-wide energy consumption is monitored via the RAPL interface. The energy efficiency is expressed in IPJ, representing the total instructions executed per unit of energy consumption. Experiments span V/f levels from 1.5 GHz to 3.5 GHz. Each workload consists of 2 to 16 applications, occupying the 16 cores on the platform. The cores are grouped into three clusters: P cluster, E1 cluster, and E2 cluster. The E clusters share clock domains, while the P cluster supports per-core DVFS adjustments. At every T epochs, one application is designated as the application of interest, migrated to a different core or cluster, and its V/f level adjusted. The energy efficiency at the end of the T epochs is the *label*, while metrics from the preceding epochs and the new configuration, i.e., new mapping and new V/f level, form the *features*.

C. Feature Selection and Engineering

Accurate energy efficiency forecasting in heterogeneous processors requires a structured feature set capturing temporal, spatial, and application-level dynamics. The features represent the behavior of the application of interest (AoI) and its interactions with co-running applications, shared resources, and the broader system environment, derived from three hierarchical levels: individual application metrics, cluster-specific aggregated metrics, and system-wide energy consumption. Individual features include fine-grained PMCs of the AoI, such as instructions executed, cycles, cache accesses, cache misses, branches, and branch misses. Cache-related metrics differ based on core type; for E clusters, they primarily reflect L2 behavior, while for P cores, they capture L3 interactions. This distinction highlights core-specific contention patterns. For instance, E clusters with high L2 miss rates may incur delays from subsequent L3 misses, while P cores directly access main memory upon L3 misses, leading to differing penalties. These metrics capture both execution behavior and resource contention dynamics across the memory hierarchy.

Cluster-specific features model the aggregated behavior of co-running applications. Metrics from applications on the same cluster as the AoI form the "Meta Application 1" group, capturing *intra-cluster contention*, such as L2 cache contention in E clusters or L3 contention in the P cluster. Aggregated metrics from applications on the other two clusters form the "Meta Applications 2 and 3" groups, reflecting *cross-cluster contention*, such as competition for shared L3 caches and memory bandwidth. To model the energy component, system-wide energy consumption is included, reflecting overall energy dynamics under varying workloads. Additionally, the core type and V/f level of the hypothetical next epoch configuration are incorporated to enable predictions under dynamic scenarios, such as task migration or frequency scaling.

Features were selected based on domain knowledge and data-driven analysis, prioritizing metrics with strong correlations to energy efficiency while eliminating redundancy. Preprocessing involved scaling numerical features to zero mean and unit variance and encoding categorical features as integers. For sequence-based modeling, features were arranged into fixed-length sequences representing consecutive epochs, ensuring temporal coherence. The final epoch in the sequence always shows a different core type and V/f level configuration of the AoI. This hierarchical feature representation captures the AoI's behavior, interactions with co-running applications, and impact on system energy efficiency. It also enables the forecasting models to account for temporal dependencies, contention patterns, and dynamic resource utilization, ensuring accurate predictions under diverse and unseen runtime conditions.

D. ML Models

Energy efficiency forecasting in heterogeneous processors requires models capable of capturing temporal dependencies, resource contention patterns, and the impact of dynamic reconfigurations. Traditional approaches, such as decision trees and shallow neural networks, struggle to model these complex relationships. Sequence-based models, specifically Transformers and LSTMs, are well-suited for this task due to their ability to process temporal sequences. Transformers leverage attention mechanisms to capture long-range dependencies, while LSTMs maintain temporal coherence through recurrent connections.

The Transformer model employs an encoder architecture to process sequences of system metrics and predict future energy efficiency. The input sequence $\{\mathbf{X}_t, \mathbf{X}_{t+1}, \ldots, \mathbf{X}_{t+T}\}$, augmented with positional encodings, is passed through *Multi-Head Attention* layers to capture temporal dependencies and incorporate parameters of hypothetical configuration changes, such as core type and V/f level. A *Feed-Forward Network* (FFN) refines these features through non-linear transformations:

$$\text{FFN}(\mathbf{u}) = \text{ReLU}(\mathbf{u}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2,$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ are learnable parameters. The final encoder output is processed by a regression head to predict the energy efficiency E_{t+T+1}. Key hyperparameters, such as the number of layers, attention heads, and hidden dimensions, are fine-tuned using grid search.

The LSTM model processes sequences by maintaining a hidden state across epochs. The input sequence is passed through an LSTM network, where each cell computes:

$$\mathbf{h}_t = \sigma(\mathbf{W}_h\mathbf{X}_t + \mathbf{U}_h\mathbf{h}_{t-1} + \mathbf{b}_h),$$

with $\mathbf{h}_t$ representing the hidden state and $\mathbf{W}_h, \mathbf{U}_h, \mathbf{b}_h$ as learnable parameters. The final hidden state is passed through a regression head to predict E_{t+T+1}. LSTM hyperparameters, including the number of layers, hidden units, and dropout rates, are fine-tuned using random search. The model employs the Adam optimizer with a learning rate scheduler for efficient convergence.

Eventually, both models are trained to minimize mean squared error (MSE), with early stopping to prevent overfitting. As will be demonstrated in the Section IV, these architectures effectively model temporal and spatial dynamics, significantly outperforming traditional baselines in energy efficiency forecasting.

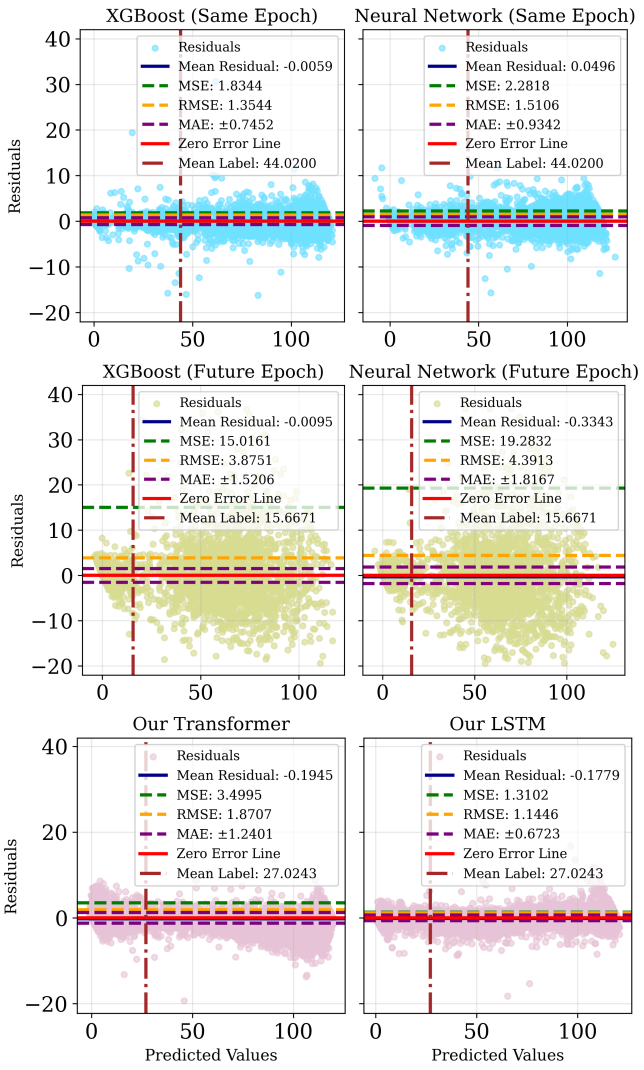


Fig. 7. Our Transformer and LSTM models demonstrate exceptional accuracy in forecasting future energy efficiency under hypothetical system configurations, with RMSE scores as low as 1.14. In contrast, simpler state-of-the-art models, while effective at predicting energy efficiency assuming no change in the immediate next epoch, fail to generalize when tasked with predicting energy efficiency for future epochs. This underscores the critical role of leveraging sequence data and advanced ML models to accurately forecast energy efficiency in dynamic and evolving scenarios.

IV. Evaluation

A. Experimental Setup and Model Architectures

All experiments and evaluations are conducted on a Linux-based machine with heterogeneous Intel® Core™ i9 12th generation Alder Lake processor. The encoder of the Transformer consists of a multi-head attention mechanism with 8 heads, each using a key dimension of 64, to capture temporal dependencies across epochs. The feed-forward network refines these features through two dense layers, the first with 128 units and ReLU activation, followed by dropout with a rate of 0.1, and the second aligning the output dimensions for residual connections. Dropout is applied after both the attention and feed-forward layers to prevent overfitting, and layer normalization ensures training stability. The final encoder output is passed through a regression head that predicts energy efficiency for the next epoch using the last time step's refined representation. The model is optimized with the *Adam* algorithm at a learning rate of 1×10^{-3} and trained for up to 100 epochs with early stopping, where validation loss is monitored with a patience of 5 epochs.

The LSTM model employs a stacked architecture with two recurrent layers. The first LSTM layer contains 64 units and outputs a sequence of hidden states, which is further condensed by a second LSTM layer with 32 units, producing only the final hidden state. Dropout regularization with a rate of 0.2 is applied after each LSTM layer to enhance generalization. The final hidden state is passed through a dense regression layer to predict the energy efficiency of the next epoch. The model is also optimized with the Adam algorithm using a learning rate of 1×10^{-3} and trained with the same early stopping criteria as the Transformer.

Both models are trained using a mean squared error (MSE) loss function, with the a dataset of 1.81 million samples, divided into subsets of training (70%), validation (15%) and testing (15%). After experimenting with multiple sequence lengths, we empirically opt for 5-epoch long sequences. The label of the model is the IPJ of the next immediate epoch, computed in millions of instructions per Joule.

B. Model Performance Comparison

As discussed in Section I and Section II-B, state-of-the-art resource management techniques utilizing machine learning focus on predicting the impact of hypothetical configurations on individual system metrics, such as performance, power, or energy. None of these approaches target energy efficiency explicitly, nor do they consider the prediction of future system states on real heterogeneous processors. To evaluate our models, we selected the two most-commonly used machine learning techniques for resource management: Neural Networks (NN) and Boosted Decision Trees (XGB). To ensure a fair comparison against our two models, we train the NN and XGB models under two distinct scenarios.

First, we trained the NN and XGB models to predict the energy efficiency of the *same* epoch using the same features as our models. Second, we trained them to predict the energy efficiency of the *future* (next) epoch, effectively simulating a sequence length of two epochs by using features from the current epoch to predict the next one. Figure 8 presents the residual distributions (actual minus predicted values) for each model, along with the MSE, RMSE, and the means of the target label and residuals.

When tasked with predicting energy efficiency for the *same* epoch, NN and XGB models demonstrated high accuracy, achieving RMSE scores of 1.51 and 1.354, respectively. However, these models struggled to predict energy efficiency for the *future* epoch in scenarios with significant temporal variance, as outlined in Section II-B. This limitation underscores their inability to effectively capture temporal dependencies and adapt to dynamic future system behaviors.

In contrast, our proposed sequence-based models—the Transformer and LSTM—excelled in the *future* task. For forecasting the energy efficiency of the *future* epoch, they achieved exceptionally low RMSE scores of 1.87 and 1.14, respectively, with mean residuals close to zero, demonstrating excellent model fitting.

C. Evaluation of Model Generalization

To evaluate its generalization ability to unseen runtime conditions, we tested our Transformer model on experiments that are *excluded from the training set*. The experiment shown in Fig. 8 features 14 applications running in parallel. Among these, we selected the same four applications in Fig. 3: *radix*, *dedup*, *ocean_ncp* and *lu_ncb*, which are known to exhibit diverse temporal sensitivity patterns. Figure 8 illustrates the forecasted and actual energy efficiency values for this scenario, evaluated every five epochs. The results confirm that our models successfully generalize to unseen contention scenarios, maintaining high prediction accuracy even for *ocean_ncb*, which previously showed recurrent energy efficiency spikes throughout its execution. Most importantly, even in the very few instances when our

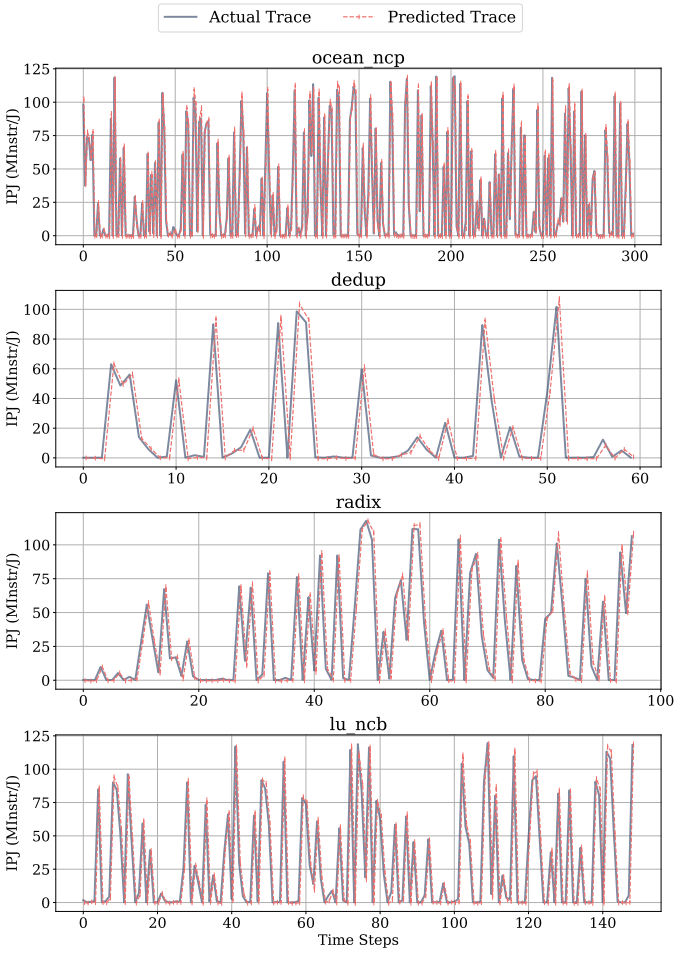


Fig. 8. Actual vs. **forecasted** energy efficiency traces of four applications while running in parallel along with 10 other background applications, in an experiment that is **unseen** during training. Every 5 epochs, the application is migrated and the V/f level of its assigned core is changed. Our Transformer model successfully captures the phase-level sensitivities to these changes along with the contention effects on the chip, delivering a very high prediction accuracy at runtime and enabling robust proactive resource management.

model mispredicted the absolute value of the label, it *never* predicted an *opposite trend* to the actual value in any of the applications across the evaluation experiments, i.e., predicting an increase in energy efficiency when it actually decreases. This demonstrates the robustness of our model architectures in capturing complex temporal and spatial dependencies inherent in heterogeneous multi-core processors. Models with such capabilities will open the door for smarter proactive resource management at runtime that can effectively optimize the energy efficiency of modern computing systems.

D. Training and Inference Overhead

While the proposed Transformer and LSTM models require longer training times compared to classical models such as simpler neural networks and XGB, this overhead is acceptable, as training is performed at design time. Once trained, the models are deployed on the target heterogeneous processor platform and used for runtime inference. Inference is offloaded to Intel's oneDNN (Deep Neural Network Library), which accelerates neural network computations by leveraging hardware-optimized routines. On our target processor, the average inference time for a single sequence evaluation is 94 ms. However, as the models utilize sequence data, we employ batch inference to process multiple sequences in parallel. This optimization reduces the average inference time drastically to only 1.82 ms per sequence. This represents less than 2% of the typical 100 ms epoch length, ensuring that the forecasting models impose minimal overhead and are well-suited for real-time energy efficiency optimization in clustered heterogeneous processors.

V. Related Work

A significant body of research has investigated application sensitivity to processor and system characteristics. Early works such as [11] proposed neural network models to optimize application-to-core mappings for minimizing energy consumption, leveraging design-time application profiles. Similarly, [12] used linear regression modelsto guide runtime application mapping decisions based on these profiles. While these techniques have considered the contention effects on the processor, they have relied on coarse-grained, whole-execution profiles, missing the finer optimization opportunities available through phase-level analysis, as demonstrated previously in Section II.

The importance of phase-level sensitivity was first highlighted in [13], [14], which demonstrated the benefits of adaptive application-to-core mapping adjustments at runtime. Although validated on simulated heterogeneous processors, these studies established the foundation for subsequent approaches. For example, [7] applied imitation learning on a real heterogeneous big.LITTLE platform to optimize performance per watt, using LLVM-based benchmarking to generate training data. Similarly, [5], [15] leveraged LLVM instrumentation to train classifiers that identify phase-level Pareto-optimal configurations at runtime. Compiler-based techniques were also explored in [6], where Java bytecode analysis via the Soot framework triggered runtime configuration changes. In [16], Linux kernel-level tracepoint instrumentation was employed to predict phase-level energy-efficient configurations. However, all these methods depend on source code access and recompilation, making them unsuitable for scenarios where source code is unavailable or recompilation is infeasible.

To address these limitations, performance counters have emerged as a non-intrusive alternative for runtime data collection without requiring source code access. In [9], phase-level performance counter data was used to train machine learning models for predicting application sensitivities to V/f level changes, enabling smart and proactive DVFS at runtime. Similar methods in [17], [18] predicted phase-level performance sensitivities for task migration. However, these works have been validated only in simulated environments, focusing solely on sensitivity to V/f levels or single core types. Furthermore, these methods assume that sensitivity remains static in the immediate subsequent epoch, a limitation demonstrated in Section II.

Recognizing this limitation, [19] introduced LSTM-based forecasting for power consumption. However, their approach was validated only on simulated homogeneous processors, limiting its applicability to real-world heterogeneous systems. In contrast, our proposed EffiCast methodology introduces a novel forecasting approach to predict the energy efficiency sensitivity of applications to both core types and V/f levels on real heterogeneous processors. This approach not only accounts for complex phase-level and contention dynamics but also generalizes to unseen runtime conditions.

VI. Conclusion

This paper introduced EffiCast, a novel methodology for forecasting energy efficiency in clustered heterogeneous multi-core processors using Transformers and LSTMs. Unlike simpler state-of-the-art techniques, our models achieved exceptional accuracy capturing complex temporal and resource contention dynamics. Deployed on real hardware with Intel's oneDNN acceleration, the models demonstrated minimal runtime overhead, with batch inference reducing per-sequence latency to a negligible 1.82 ms. These results validate the integration of energy efficiency forecasting into proactive resource management frameworks, enabling smarter task mapping and DVFS strategies for energy-constrained systems.

REFERENCES

[1] B. Kocot, P. Czarnul, and J. Proficz, "Energy-aware scheduling for high-performance computing systems: A survey," *Energies*, vol. 16, no. 2, 2023. [Online]. Available: https://www.mdpi.com/1996-1073/16/2/890

[2] "Hybrid architecture (code name alder lake)." [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/hybrid-architecture.htm

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2008.

[4] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Int. Symp. Computer Architecture (ISCA)*, 1995.

[5] U. Gupta, C. A. Patil, G. Bhat, P. Mishra, and U. Y. Ogras, "Dypo: Dynamic pareto-optimal configuration selection for heterogeneous mpsocs," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, sep 2017. [Online]. Available: https://doi.org/10.1145/3126530

[6] J. C. R. Da Silva, L. Leão, V. Petrucci, A. Gamatié, and F. M. Q. a. Pereira, "Mapping computations in heterogeneous multicore systems with statistical regression on program inputs," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 6, oct 2021. [Online]. Available: https://doi.org/10.1145/3478288

[7] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "Dynamic resource management of heterogeneous mobile platforms via imitation learning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2842–2854, 2019.

[8] J. S. Shim, B. Han, Y. Kim, and J. Kim, "Deeppm: Transformer-based power and performance prediction for energy-aware software," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 1491–1496.

[9] M. Rapp, M. B. Sikal, H. Khdr, and J. Henkel, "Smartboost: Lightweight ml-driven boosting for thermally-constrained many-core processors," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 265–270.

[10] "perf: Linux profiling with performance counters," Jun 2009. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[11] A. Edun, R. Vazquez, A. Gordon-Ross, and G. Stitt, "Dynamic scheduling on heterogeneous multicores," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1685–1690.

[12] E. Shamsa, A. Kanduri, P. Liljeberg, and A. M. Rahmani, "Concurrent application bias scheduling for energy efficiency of heterogeneous multi-core platforms," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 743–755, 2022.

[13] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," in *Proceedings of the 46th Annual Design Automation Conference*, ser. DAC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 927–930. [Online]. Available: https://doi.org/10.1145/1629911.1630149

[14] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 213–224.

[15] S. K. Mandal, G. Bhat, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "An energy-aware online learning framework for resource management in heterogeneous platforms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 3, may 2020. [Online]. Available: https://doi.org/10.1145/3386359

[16] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt, "Sparta: Runtime task allocation for energy efficient heterogeneous manycores," in *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016, pp. 1–10.

[17] M. B. Sikal, H. Khdr, M. Rapp, and J. Henkel, "Machine learning-based thermally-safe cache contention mitigation in clustered manycores," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[18] M. Rapp, A. Pathania, T. Mitra, and J. Henkel, "Neural network-based performance prediction for task migration on s-nuca many-cores," *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1691–1704, 2021.

[19] M. Sagi, M. Rapp, H. Khdr, Y. Zhang, N. Fasfous, N. A. Vu Doan, T. Wild, J. Henkel, and A. Herkersdorf, "Long short-term memory neural network-based power forecasting of multi-core processors," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1685–1690.

Joint Cutting for Hybrid Schrödinger-Feynman Simulation of Quantum Circuits

Laura S. Herzog[†], Lukas Burgholzer[†§], Christian Ufrecht[‡], Daniel D. Scherer[‡], Robert Wille[†§¶]
[†] Chair for Design Automation, Technical University of Munich, Germany
[‡]Fraunhofer Institute for Integrated Circuits IIS, Nuremberg, Germany
[§]Munich Quantum Software Company GmbH, Garching near Munich, Germany
[¶]Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria

***Abstract*—Despite the continuous advancements in size and robustness of real quantum devices, reliable large-scale quantum computers are not yet available. Hence, classical simulation of quantum algorithms remains crucial for testing new methods and estimating quantum advantage. Pushing classical simulation methods to their limit is essential, particularly due to their inherent exponential complexity. Besides the established *Schrödinger-style* full statevector simulation, so-called *Hybrid Schrödinger-Feynman* (HSF) approaches have shown promise to make simulations more efficient. HSF simulation employs the idea of "cutting" the circuit into smaller parts, reducing their execution times. This, however, comes at the cost of an exponential overhead in the number of cuts. Inspired by the domain of Quantum Circuit Cutting, we propose an HSF simulation method based on the idea of "joint cutting" to significantly reduce the aforementioned overhead. This means that, prior to the cutting procedure, gates are collected into "blocks" and all gates in a block are jointly cut instead of individually. We investigate how the proposed refinement can help decrease simulation times and highlight the remaining challenges. Experimental evaluations show that "joint cutting" can outperform the standard HSF simulation by up to a factor $\approx 4000\times$ and the Schrödinger-style simulation by a factor $\approx 200\times$ for suitable instances. The implementation is available at https://github.com/cda-tum/mqt-qsim-joint-cutting.**

***Index Terms*—quantum computing, classical simulation, hybrid Schrödinger-Feynman, joint cutting, circuit cutting**

I. Introduction

As the availability of reliable large-scale, fault-tolerant quantum computers is still pending, classical simulation of quantum circuits remains a central tool for quantum computing research—including developing and testing quantum algorithms as well as comparing classical computers against current quantum hardware. Such simulations are inherently challenging due to the exponential scaling of the statevectors' dimension with a growing number n of qubits. The standard approach for quantum circuit simulation is the so-called *Schrödinger-style* simulation which directly applies the quantum gates on the full statevector via matrix-vector multiplication. As this requires storing a vector with 2^n entries, storage capacities are quickly exhausted. Additionally, the runtimes increase heavily with growing n. Even though the exponential scaling cannot be avoided, any decrease in memory and runtime is desirable.

Hybrid Schrödinger-Feynman (HSF) [1]–[3] techniques tackle this by trading memory complexity for time complexity. This is done by partitioning large circuits into smaller subcircuits, e.g., with $\lceil n/2 \rceil$ qubits each. However, such partitioning requires "cutting" gates that connect the different partition which generates multiple, smaller, subcircuits to be simulated. With an increasing number of cuts, also the number of subcircuits grows exponentially. This count of smaller subcircuits, so-called "paths", is thus determined by the chosen quantum algorithm to be simulated—allowing faster runtimes as long as the circuit's cut induces only a reasonable number of paths.

The state-of-the-art HSF technique applies such "cuts" separately on each gate to be cut. We propose enhancing this procedure by, first, grouping the gates to be cut and, then, performing a "joint cut" on the grouped gates. Gates can be cut jointly by multiplying the respective gates and performing a Schmidt Decomposition afterwards—while this can be done automatically, one can also find analytical expressions for specific cut blocks. For certain classes of quantum circuits, the proposed method not only speeds up simulation runtimes compared to standard HSF computations but also extends the HSF simulation's applicability to instances where standard Schrödinger-style simulation would otherwise have been faster.

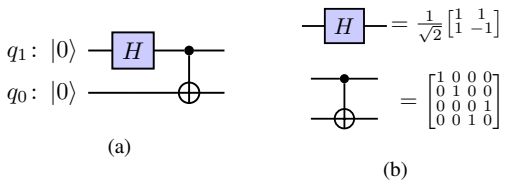
The remainder of this paper is structured as follows. In Sec. II the necessary basics about quantum computing and quantum circuit simulation as well as the concepts behind HSF simulation are detailed. Afterwards, in Sec. III, we introduce the idea of "joint cutting" and review related work. Following up on this, Sec. IV provides important details about the realization of the proposed idea along with practical considerations. Finally, the results obtained during the evaluation of the proposed approach are summarized in Sec. V and the paper is concluded in Sec. VI.

II. Background

To keep this paper self-contained, this section reviews the basic nomenclature of quantum circuits, the core ideas of quantum circuit simulation on classical computers, and the essential concepts of Hybrid Schrödinger-Feynman simulation.

A. Quantum Circuits and Schrödinger-style Simulation

An n-qubit quantum system can be described by its *statevector*, an element of a Hilbert space $\mathcal{H}_2^{\otimes n}$, where the Hilbert space of a single qubit is $\mathcal{H}_2 = \text{span}(\{|0\rangle, |1\rangle\})$. Thus, the dimension of this space scales exponentially with the number n of qubits in the considered quantum system—leading to inevitable limits regarding storing such vectors on



(a) (b)

Fig. 1: Quantum circuit (a) and the corresponding gates (b) for a Bell state preparation.

classical machines. In Dirac notation, a state $|\psi\rangle$ residing in this space can be written as

$$|\psi\rangle = \sum_{i\in\{0,1\}^n} \psi_i \, |i\rangle \,, \tag{1}$$

with the amplitudes $\psi_i \in \mathbb{C}$ and $i = (i_{n-1}, ..., i_0) \in \{0,1\}^n$. The qubits' state can be manipulated by applying quantum gates, which are described as unitary operators acting on the Hilbert space. Representing the operator in the computational basis, the output state after application of the corresponding gate can be determined by applying matrix-vector multiplication. Performing this procedure of consecutive matrix-vector multiplications on a classical computer is the core idea of *Schrödinger-style* simulation. We refer to [4] for a broader review.

Example 1. *Consider the quantum circuit shown in Fig. 1a. The computation starts with the all-zero statevector for two qubits with $2^2 = 4$ entries, i.e.,*

$$|00\rangle = |0\rangle \otimes |0\rangle = [1,0]^T \otimes [1,0]^T = [1,0,0,0]^T. \tag{2}$$

Following the circuit, a Hadamard and a CNOT gate are applied, whose matrix representations are shown in Fig. 1b. Directly performing the consecutive matrix-vector multiplication on $|00\rangle$ leads to the state

$$\big(\mathrm{CNOT} \cdot \big((H \otimes I)\,|00\rangle\big)\big) = \frac{1}{\sqrt{2}} \cdot [1,0,0,1]^T, \tag{3}$$

with I being the 2×2 identity matrix. The result corresponds to the well-known Bell state $|\Phi\rangle^+ = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

Classical simulation of quantum circuits, as reviewed above, can be performed by employing different data structures for representing the exponential number of amplitudes in the statevector. Directly following the treatment in Ex. 1, one can use plain arrays to store the statevector, matrix representations of gates, and for performing matrix-vector multiplication. Alternative structures have been investigated to cope with the exponential complexity. Examples include *tensor networks* [5]–[8] which emerged from the field of condensed matter physics as well as *Decision Diagrams* (DDs) originating from the *Electronic Design Automation* (EDA) community [9]–[15].

In addition to various data structures, alternative simulation schemes to standard Schrödinger-style simulation exist, as its major issue is storing and handling the exponentially large statevector. A prominent method that tackles this weakness, *Hybrid Schrödinger-Feynman* (HSF) simulation, is reviewed next.

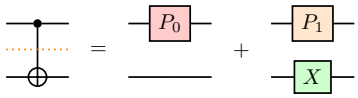


Fig. 2: Bipartite representation of the CNOT gate. The orange, dotted line indicates the "cut".

B. Hybrid Schrödinger-Feynman Simulation

HSF simulation [1]–[3] aims to reduce the inherent exponential memory complexity involved in classically simulating quantum circuits—at the expense of exponential runtime. The idea of HSF simulation is to horizontally partition the circuit into, for instance, two subcircuits with a similar number of qubits. This reduces the memory complexity, e.g., from $\mathcal{O}(2^n)$ to $\mathcal{O}(2^{\lceil n/2 \rceil})$, assuming a partitioning into two subcircuits with $\approx n/2$ qubits each. Since a fraction of gates usually acts across both partitions, this procedure requires "cutting" those connecting gates. This needs a factorized representation of the gate that ensures its constituents only act locally on their partition. The following example illustrates the idea.

Example 2. *Consider the CNOT gate as an example. Finding a bipartite representation of the CNOT gate can be easily done by factoring out properly, i.e.,*

$$\mathrm{CNOT} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = P_0 \otimes I + P_1 \otimes X.$$

An illustration of this factorization is provided in Fig. 2. Thus, a CNOT gate can be directly decomposed into two terms composed of local unitaries only.

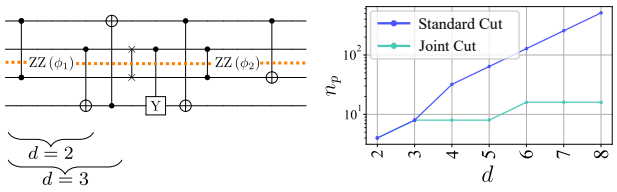
To find such bipartite representations more generally, one can perform a *Schmidt Decomposition* [4] of the corresponding unitary. As will be explained in more detail in Sec. IV, the Schmidt Decomposition allows to find representations of the form

$$A = \sum_{m=0}^{r-1} \sigma_m \, X_m \otimes Y_m, \tag{4}$$

for arbitrary operators $A : \mathcal{H}_2^{\otimes n} \to \mathcal{H}_2^{\otimes n}$. The matrix of X_m is of size $2^{n_a} \times 2^{n_a}$ while Y_m is of size $2^{n_b} \times 2^{n_b}$ such that $n_a + n_b = n$ holds. The value r is denoted as the *Schmidt-rank* and σ_m are the singular values.

Thus, one can find a bipartition to perform a cut for general n-qubit gates. Each term of the Schmidt-decomposed gate constitutes a pair of circuits to be simulated separately—each pair being a "path" in the overall simulation. If all gates connecting the two partitions are cut, one receives a set of smaller subcircuits, such that their memory complexity can be reduced, e.g., from 2^n to $2^{\max(n_a, n_b)}$. This reduced memory complexity then also reduces the overall runtime. The number of elements in this set, however, scales exponentially with the number of gates being cut. This leads to an overhead in time complexity, which can partly be remedied by parallel simulation of the numerous paths. The final result is reconstructed by applying the Kronecker product between the subcircuits per simulation and adding up all contributions.

The HSF technique is named for its hybrid approach, combining Schrödinger-style simulation (using matrix-vector multiplication for subcircuits) with Feynman-style simulation



(a) (b)

Fig. 3: (a) An exemplary circuit of which different depths d are considered. For $d = 2$ only the two leftmost gates are included and so on. The dotted, orange line indicates the location of the cut. (b) For different depths d, the number of paths n_p increases more rapidly with standard cutting compared to joint cuts, which saturate. The steeper slope from $d = 3$ to $d = 4$ for standard cutting is due to the SWAP gate's Schmidt rank $r = 4$, whereas the others have $r = 2$.

that explores different "paths". Although a full Feynman-style simulation, i.e., cutting *all* multi-qubit gates, is computationally impractical, the hybrid method has proven quite effective.

III. Motivation and General Idea

The main strength of the HSF simulation is to reduce the memory complexity by partitioning the circuit into smaller parts. However, this comes with a cost: The more gates are cut during the partitioning, the more paths have to be simulated. Generally, the number n_p of paths scales exponentially with the number of cuts. If m gates are cut, one has $n_p = \prod_{i=0}^{m-1} r_i$ paths where r_i is the Schmidt-rank of each cut gate. If all gate decompositions have the same rank r, one can simplify this to $n_p = r^m$. Thus, not every quantum circuit is suitable for being simulated with HSF approaches: Even with parallelization of the different paths, the exponentially increasing n_p quickly renders the runtime prohibitive. Hence, the HSF simulation offers advantages for shallow circuits and circuits that can be partitioned into weakly connected sections—allowing for a sweet spot in the trade-off between reduced memory usage and the increased number of paths.

However, in this work we show that the full potential of HSF simulation has not been reached yet. Reducing n_p is possible by going beyond the naive approach of cutting each gate separately. In this work, we propose to perform "joint cuts" by forming suitable groups of gates in the circuit first and cutting them jointly with a Schmidt Decomposition. After illustrating how this procedure can be exploited in principle, the connections of our work and the related field of Quantum Circuit Cutting (QCC) [16]–[22] will be briefly reviewed.

A. Reducing the Number of Paths with "Joint Cutting"

As mentioned above, individually applying cuts to each gate that connects multiple partitions directly results in an exponentially growing number n_p of paths. While this exponential scaling cannot be completely avoided, there is some flexibility in the way gates are cut. For instance, if several gate cuts can be grouped, i.e., *jointly* cut, the number n_p of paths can be reduced considerably. An example illustrates the effect.

Example 3. *Consider the circuit to be simulated as shown in Fig. 3a and assume a partitioning that leads to a subcircuit with qubits q_0 and q_1 as well as one with qubits q_2 and q_3. In a straightforward application of HSF simulation, one performs a Schmidt decomposition on every single gate on the cut, such that n_p is the product of all separate Schmidt-ranks. Instead, we can combine the gates first and then cut them jointly by performing a Schmidt Decomposition of the resulting unitary. The behavior of n_p in these cases is summarized in Fig. 3b, where n_p is plotted for increasing depths d of the example circuit from Fig. 3a. This clearly shows the potential of the "joint cutting" approach for which the number of paths saturates at $n_p = 2^{2 \cdot 2} = 16$ while the state-of-the-art cutting scheme scales exponentially with d.*

While this example was artificially created to demonstrate the potential benefits, our experimental evaluations, which are summarized later in Sec. V, show the practical advantage of the proposed "joint cutting" scheme.

The number n_p of paths is reminiscent of a corresponding quantity within the field of *Quantum Circuit Cutting* (QCC), which is briefly reviewed in the next section.

B. Related Work

An advantageous aspect of (joint) cutting for quantum circuit simulation is that the *Singular Value Decomposition* (SVD) can be found for any matrix, and therefore the Schmidt Decomposition (Sec. IV-A) can always be constructed. Thus, the cuts can be performed fully automatically, even though it is also possible to find analytical decompositions as demonstrated in Sec. IV-D. HSF simulation techniques enjoy further freedom since the decomposition does not need to correspond to quantum operations, hence do not need to be unitary, such as the projectors P_0, P_1 from Ex. 2. Changing the point of view regarding the treatment of algorithms on real quantum devices, cutting can be performed as well, but in a more constrained manner.

QCC aims to partition quantum circuits such that they can be run on smaller quantum computers. Hence, QCC and HSF approaches stem from the same core idea: Circumventing the effects of limited memory—be it RAM on a classical computer or the number of qubits in a quantum computer. Additionally, QCC can also help to reduce the impact of noise on current NISQ devices [23]. Though both fields are related, decomposing gates in a QCC framework differs notably from the Schmidt decomposition applied in HSF simulations. In contrast to the aspects mentioned above, finding a decomposition for individual or combined gates cannot be done automatically and necessitates detailed, manual examination on a case-by-case basis.

More formally, in QCC the quantum channel of the gate is decomposed, in contrast to HSF simulation, where we decompose the matrix representing the gate. More specifically, in QCC, the decomposition takes the form of a *Quasi-Probability Decomposition* (QPD) $\mathcal{W} = \sum_i a_i \mathcal{F}_i$. Each of the $\mathcal{F}_i$ is a bipartite channel, where operations are applied independently to each partition without connecting them, similar to the illustration in Fig. 2. The QPD is realized on a quantum computer by sampling the operation $\mathcal{F}_i$ with probability proportional to $|a_i|$ and recombining the results by classical postprocessing. This procedure comes with a *sampling overhead* which scales exponentially with the number of cuts. Note that the number of $\mathcal{F}_i$'s is reminiscent of the number n_p of paths in the

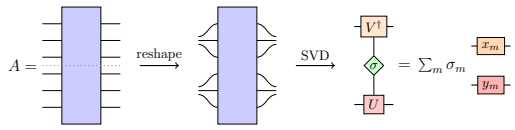


Fig. 4: Illustration of reshaping the matrix of A and performing an SVD.

HSF simulation and also scales exponentially with the number of cuts in the case of QCC. An endeavor in QCC is to reduce both the number of $\mathcal{F}_i$'s as well as the aforementioned sampling overhead. Even though the exponential scaling cannot be avoided, it can be remedied to some extent by introducing "joint cutting" [21], [24]–[26] for QCC. Similar to the proposed "joint cutting" for HSF simulation, in QCC, multiple gates are combined and, while exploiting the resulting structure, more beneficial decompositions can be found.

IV. Technical Details

In the following, we will elaborate on the ideas proposed above and illustrated in Ex. 3, i.e., how the Schmidt Decomposition is performed on general (blocks of) gates and why we can reach a saturation in n_p. Furthermore, we point out that the overhead from the Schmidt Decomposition can become significant and how "joint cutting" can be used in practice.

A. General Schmidt Decomposition

The core idea of the Schmidt Decomposition is to perform a *Singular Value Decomposition* (SVD) and to rotate the basis as illustrated in Fig. 4. As indicated in the figure, one can reshape the qubit wires (interpreted as legs of the corresponding tensor) according to the cut (orange dotted line). Afterwards, the SVD is performed which constitutes a bipartite representation. Mathematically speaking, each quantum gate is an operator acting on the qubits' Hilbert space. Given an operator $A: \mathcal{H}_2^{\otimes n} \to \mathcal{H}_2^{\otimes n}$ with n qubits, it can be written as

$$A = \sum_{i,j \in \{0,1\}^n} A_{i,j} \left|i\right\rangle \left\langle j\right|, \tag{5}$$

where $i = (i_{n-1}, ..., i_0)$ and $j = (j_{n-1}, ..., j_0)$. The coefficients $A_{i,j}$ can be regarded as a 2^{2n}-dimensional object. We want to decompose the unitary into partition a and b with the indices $i^a = (i_{n-1}, ..., i_{l+1})$, $i^b = (i_l, ...i_0)$ and equivalently for j^a and j^b. Thus, the cut is performed between qubit l and $l+1$. This requires reshaping and matricization, i.e.,

$$A = \sum_{\mathbf{i}_{\leq l}, \mathbf{i}_{>l}} \tilde{A}_{\mathbf{i}_{\leq l}, \mathbf{i}_{>l}} \left|i^a\right\rangle \left\langle j^a\right| \otimes \left|i^b\right\rangle \left\langle j^b\right|, \tag{6}$$

with $\mathbf{i}_{\leq l} = (i^b, j^b)$ and $\mathbf{i}_{>l} = (i^a, j^a)$ as well as $\tilde{A}_{\mathbf{i}_{\leq l}, \mathbf{i}_{>l}}$ is a relabeled $A_{i,j}$. Now, the SVD can be applied such that

$$\tilde{A}_{\mathbf{i}_{\leq l}, \mathbf{i}_{>l}} = \sum_{m=0}^{r-1} U_{\mathbf{i}_{\leq l}, m} \sigma_m V^*_{m, \mathbf{i}_{>l}}, \tag{7}$$

with the *Schmidt-rank* r. The U and V^* can be absorbed in the basis as

$$X_m = \sum_{\mathbf{i}_{>l}} V^*_{m, \mathbf{i}_{>l}} \left|i^a\right\rangle \left\langle j^a\right| \tag{8}$$

$$Y_m = \sum_{\mathbf{i}_{\leq l}} U_{\mathbf{i}_{\leq l}, m} \left|i^b\right\rangle \left\langle j^b\right|, \tag{9}$$

leading to a bipartite representation of the original matrix of A shown in Eq. 4.

B. Theoretical Guarantee for Lower n_p

In general, it is known that the Schmidt-rank of a matrix is limited by its dimension. This means that the matrix of an operator $A: \mathcal{H}_2^{\otimes n} \to \mathcal{H}_2^{\otimes n}$ of a block with n qubits which is split into partitions with n_a, n_b qubits, respectively, can have a Schmidt-rank at most $\tilde{r} = \min(2^{2n_a}, 2^{2n_b})$ [27]. This behavior is the explanation for the saturation of n_p for the "joint cutting" in Ex. 3: While this bound applies individually to each cut gate when applied consecutively, it does not hold collectively for all gates. In contrast, for joint gate cutting, the limit applies to the entire block—leading to the aforementioned saturation of $2^{2 \cdot 2}$. Therefore, "joint cutting" is guaranteed to reduce n_p for deep enough blocks with limited dimensions.
To illustrate this in mathematical terms, consider a sequence of operators $\mathcal{N} = \{A_k\}_{k=0}^{K-1}$ where $A_k: \mathcal{H}_2^{\otimes n} \to \mathcal{H}_2^{\otimes n}$. They are cut into partitions with n_a, n_b qubits respectively. The "joint cutting" is guaranteed to reduce n_p as soon as

$$n_p^t = \prod_{k=0}^{K-1} r_k > \tilde{r} \geq n_p^j \tag{10}$$

holds, where r_k denotes the Schmidt-rank of the matrix of A_k. The left-hand side comes from multiplying up all Schmidt-ranks within $\mathcal{N}$ for the standard cutting, leading to n_p^t paths. On the other hand, the maximum number of "joint cutting" paths, n_p^j, is upper bounded by $\tilde{r}$, i.e., the smaller dimension of the reshaped matrix on which the SVD is performed (Eq. 7).

One should note, however, that in Ex. 3, already before the saturation of n_p, the "joint cutting" reduces n_p in comparison to the standard cutting. Thus, the above guarantee does not need to be exhausted to make "joint cutting" for HSF simulations beneficial.

C. Overhead for "Joint Cutting"

Joining gates into blocks for "joint cutting" comes with a cost that has to be drawn into consideration regarding the usefulness of the proposed "joint cutting" technique. In general, merging a block of gates into one requires consecutive matrix multiplications first and afterwards a Schmidt Decomposition. The first usually has the asymptotic time complexity $\mathcal{O}(D^s)$ with $2 \leq s \leq 3$ and the SVD within the latter $\mathcal{O}(D^3)$. Here, $D = 2^k$ for a k-qubit block, and therefore including too many qubits in a block in relation to the whole circuit can make "joint cutting" ineffective—unless analytical decompositions can be found (Sec. IV-D).

Therefore Ex. 3 may have a reduced number of paths n_p but the preprocessing would dominate the total simulation runtime. Strictly speaking, the preprocessing is more costly than direct Schrödinger simulation, as the latter primarily involves matrix-vector multiplication ($\mathcal{O}(D^2)$). As a consequence, "joint cutting" can only be useful if utilized with care: Joining only a few gates in a large circuit can severely decrease the number of paths n_p. If small enough blocks are chosen, one receives the best of both worlds: A reduction in n_p as well as negligible preprocessing overhead.

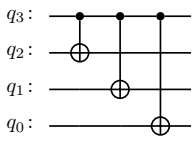


Fig. 5: A "cascade" of CNOT gates.

D. "Joint Cutting" in Practice

In practice, it is recommended to consider quantum circuits following a beneficial structure for "joint cutting". One such structure is "cascades" of two-qubit gates.

Example 4. *Consider Fig. 5 in which an example cascade of CNOT gates is displayed. Referring to the decomposition of the CNOT gate in Ex. 2, one can deduce the decomposition of the cascade of three CNOT gates* $C_{\mathrm{CNOT},3}$ *as*

$$C_{\mathrm{CNOT},3} = P_0 \otimes I \otimes I \otimes I + P_1 \otimes X \otimes X \otimes X. \qquad (11)$$

Thus, the Schmidt-rank remains $r = 2$*, such that a joint cut of this structure would be much smarter than standard cutting with* $n_p^t = 2^3$.

This structure is also beneficial for different kinds of gates such as, for instance, CZ or RZZ gates—which occur in many quantum algorithms.

V. Experimental Evaluation

To evaluate the performance of the proposed "joint cutting" method for HSF simulations, we built upon Google's qsim package [28], a high-performant array-based quantum simulator used for cross-entropy benchmarking in [29]. The "joint cutting" implementation is open-source and available at https://github.com/cda-tum/mqt-qsim-joint-cutting. All evaluations were performed on a machine with a 16-core AMD Ryzen Threadripper PRO 5955WX CPU and 128 GB RAM. To keep the computation times reasonably low while still considering instances of relevant size, only a fraction of the total amplitudes (10^6) was computed for all considered methods.

As a case study, we look at quantum algorithms for solving *Quadratic Unconstrained Binary Optimization* (QUBO) problems, which, at the time of writing, arguably constitute the largest area of research when it comes to applications of quantum computing. In particular, we consider the *Quantum Approximate Optimization Algorithm* (QAOA) for solving the Max Cut problem, as it has been shown that any QUBO problem can be reduced to a weighted Max Cut instance [30]–[33]. The circuits consist of multiple alternating so-called *problem* and *mixer* layers, for which one uses RZZ entangling gates as well as single-qubit RX rotation gates, respectively. RZZ gates mutually commute, which gives a lot of freedom to reorder the gates such that grouping and, therefore, "joint cutting" can be performed—see Fig. 6 for an example.

To control the number of gates to cut for the HSF simulation, we considered graphs with two partitions of nodes with roughly the same size and place the cut between the partitions. Nodes are connected with probability p_{intra} within the partition and with, usually lower, probability p_{inter} between the partitions. Note that an original, denser, problem graph could

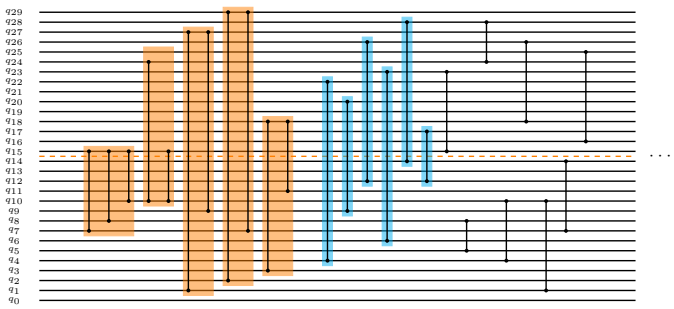


Fig. 6: Examples of RZZ gates of a problem layer in QAOA from a graph with 15 qubits per partition (q30-1). The orange dotted line denotes the cut and the orange shaded gates are the utilized blocks. Blue shaded gates are cut separately. Only a few of the uncut gates are displayed without coloring.

be shrunk to an instance that is easier to execute [34]. Given the cut location, we use a brute force algorithm to reassemble cascades of RZZ gates which can be cut jointly. To keep the implementation more general, the joint cuts were performed numerically, even though analytical expressions can be found for "cascade" structures as shown in Ex. 4.

Tab. I summarizes the results of evaluating qsim's Schrödinger Simulation, its standard, state-of-the-art HSF Simulation, and the proposed Joint HSF Simulation on various single-layer QAOA instances. Detailed specifications of the circuits are shown in Tab. II. As expected, the Schrödinger simulation's runtime mainly depends on the number of qubits—approximately doubling with every qubit added. However, for both the standard and joint HSF simulation the runtimes are heavily influenced by the number of paths. As implied in Sec. III, if the sweet spot is missed, the exponentially growing number of paths can spoil the efficiency of standard HSF simulations, making it slower than Schrödinger simulation. The evaluated QAOA instances belong to this class for which HSF simulation is, in its original form, not a recommendable technique. However, the proposed "joint cutting" can reduce the number of paths to such an extent that it can speed up the standard HSF simulation by up to a factor $\approx 4000\times$, which, in turn, allows to outperform the Schrödinger simulation by up to a factor of $\approx 200\times$. Thus, the proposed "joint cutting" enables HSF techniques to be useful for instances for which the standard HSF procedure fails. This is also made possible by keeping the preprocessing costs for "joint cutting" low, as emphasized in Sec. IV-C. Notably, the proposed method is *always* faster than the standard HSF simulation since cascades could be found in all instances.

Finally, note that "joint cutting" can be applied to other classes of circuits as well. For instance, one could simulate deeper circuits for quantum many-body dynamics as those in [35] or join CZ or iSWAP gates in shallow instances of Google's supremacy circuits [36]. We tested the latter but due to page limitations, these results are relegated to the aforementioned GitHub repository.

VI. Conclusions

In conclusion, this work presented a method for enhancing HSF simulation, enabling it to be faster than Schrödinger

TABLE I: Runtimes of QAOA circuits. For the runtimes, the first line is the full time with preprocessing, and the second line is only the simulation itself. Note that the preprocessing not only contains the Schmidt Decomposition and construction of paths for the "joint cutting" but also gate fusion in both cases [37], [38]. The mean of those runtimes is displayed. In brackets, one can see the standard deviation from the mean of five runs. The column S/J shows the total *S*chrödinger time divided by the full "joint cutting" time, and T/J the same but for the full time of standard cutting divided by the full "joint cutting" time. If the standard cutting is timed out, the lower bound of T/J is given. The bold entries display those in which the proposed method performs better. For all computations, the first 10^6 amplitudes were computed.

	Schrödinger	Standard HSF		Proposed Joint Cutting HSF		Performance Ratios	
Circuit	Runtime (s)	Runtime (s)	# Paths	Runtime (s)	# Paths	S/J	T/J
q30-1	43.177 (0.149) 42.915 (0.150)	118.022 (2.041) 118.022 (2.041)	2^{17}	0.618 (0.021) 0.618 (0.021)	2^{11}	**69.857**	**190.950**
q30-2	45.292 (0.024) 45.029 (0.026)	timed out (1 h)	2^{26}	1.480 (0.054) 1.444 (0.055)	2^{12}	**30.604**	$\geq$ **2432.544**
q30-3	46.280 (0.051) 46.018 (0.051)	timed out (1 h)	2^{29}	49.003 (1.648) 48.991 (1.646)	2^{17}	0.944	$\geq$ **73.465**
q31-1	91.956 (0.120) 91.435 (0.122)	308.671 (4.552) 308.671 (4.552)	2^{18}	0.389 (0.050) 0.379 (0.049)	2^{10}	**236.188**	**792.821**
q31-2	97.372 (0.056) 96.851 (0.056)	timed out (1 h)	2^{28}	13.874 (0.122) 13.863 (0.122)	2^{15}	**7.018**	$\geq$ **259.473**
q31-3	99.860 (0.051) 99.339 (0.050)	timed out (1 h)	2^{34}	6.670 (0.152) 6.620 (0.146)	2^{14}	**14.971**	$\geq$ **539.726**
q32-1	201.494 (0.039) 200.457 (0.041)	timed out (1 h)	2^{24}	0.848 (0.012) 0.822 (0.005)	2^{11}	**237.692**	$\geq$ **4246.730**
q32-2	202.769 (0.060) 201.728 (0.063)	timed out (1 h)	2^{25}	1.605 (0.008) 1.564 (0.007)	2^{12}	**126.356**	$\geq$ **2243.348**
q32-3	209.714 (0.042) 208.676 (0.042)	timed out (1 h)	2^{30}	3.219 (0.036) 3.165 (0.037)	2^{13}	**65.147**	$\geq$ **1118.324**
q33-1	427.595 (0.102) 425.519 (0.095)	timed out (1 h)	2^{24}	7.409 (0.264) 7.402 (0.263)	2^{14}	**57.711**	$\geq$ **485.879**
q33-2	435.864 (0.183) 433.787 (0.188)	timed out (1 h)	2^{27}	34.042 (1.109) 34.031 (1.110)	2^{16}	**12.804**	$\geq$ **105.751**
q33-3	463.847 (12.571) 457.035 (3.128)	timed out (1 h)	2^{30}	16.884 (0.100) 16.843 (0.104)	2^{15}	**27.473**	$\geq$ **213.221**

TABLE II: Specifications of the QAOA circuits where the problem graphs are generated with networkx' stochastic_block_model. The column "sizes" indicates the number of vertices per partition of the problem graph, which sum up to q and have connecting edges within the partition with probability p_{intra} and between the partitions with p_{inter}. The "cut pos." denotes the qubit label after which the cut is performed (roughly after half the qubits q). "blocks + sep" counts the number of cut blocks and the remaining separate cuts (which could not be summarized in a block). "sep. cuts" counts the total separate cuts. The total number of 2-qubit gates in the circuit is given in column "# 2-qubit gates". All QAOA instances contain a single problem layer and a single mixer layer.

Circuit	q	cut pos.	# 2-qubit gates	sizes	p_{inter}	p_{intra}	blocks + sep.	sep. cuts
q30-1	30	14	172	[15,15]	0.1	0.8	5+6	17
q30-2	30	14	181	[15,15]	0.15	0.8	6+6	26
q30-3	30	14	185	[15,15]	0.17	0.8	7+10	29
q31-1	31	14	186	[15,16]	0.1	0.8	6+4	18
q31-2	31	14	197	[15,16]	0.15	0.8	8+7	28
q31-3	31	14	203	[15,16]	0.17	0.8	9+5	34
q32-1	32	15	206	[16,16]	0.1	0.8	6+5	24
q32-2	32	15	207	[16,16]	0.11	0.8	5+7	25
q32-3	32	15	214	[16,16]	0.12	0.8	7+6	30
q33-1	33	15	219	[16,17]	0.1	0.8	6+8	24
q33-2	33	15	223	[16,17]	0.11	0.8	6+10	27
q33-3	33	15	234	[16,17]	0.12	0.8	8+7	30

simulation for instances, in which the standard HSF simulation would fail. This approach is effective when circuits contain identifiable blocks (e.g., cascades), making it a valuable tool for structured circuit types. However, it is important to note that, like Quantum Circuit Cutting, HSF simulation remains best suited for specific circuit shapes, with challenges still arising for deep and dense circuits—even for the enhanced "joint cutting". While these limitations persist, our results highlight meaningful advancements in the scope and applicability of HSF techniques. Future work could explore further refinements in gate grouping for "joint cutting". For instance, in addition to regrouping the gates, adjusting the qubit order itself may help further to identify beneficial blocks that can be cut jointly.

ACKNOWLEDGMENT

This work received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001318), was part of the Munich Quantum Valley, which the Bavarian state government supports with funds from the Hightech Agenda Bayern Plus, and has been supported by the BMWK based on a decision by the German Bundestag through project QuaST, as well as by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

[1] Scott Aaronson and Lijie Chen, *Complexity-theoretic foundations of quantum supremacy experiments*, 2016. DOI: 10.48550/arXiv.1612.05903. arXiv: 1612.05903[quant-ph].

[2] Igor L. Markov *et al.*, *Quantum supremacy is both closer and farther than it appears*, 2018. DOI: 10.48550/arXiv.1807.10749. arXiv: 1807.10749[quant-ph].

[3] Zhao-Yun Chen *et al.*, "64-qubit quantum circuit simulation," *Science Bulletin*, vol. 63, no. 15, pp. 964–971, 2018. DOI: 10.1016/j.scib.2018.06.007.

[4] Michael A. Nielsen and Isaac L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 2010, ISBN: 9780511976667 Publisher: Cambridge University Press. DOI: 10.1017/CBO9780511976667.

[5] Danylo Lykov *et al.*, *Tensor network quantum simulator with step-dependent parallelization*, 2022. DOI: 10.48550/arXiv.2012.02430. arXiv: 2012.02430[quant-ph].

[6] Johnnie Gray and Stefanos Kourtis, "Hyper-optimized tensor network contraction," *Quantum*, vol. 5, p. 410, 2021. DOI: 10.22331/q-2021-03-15-410. arXiv: 2002.01935[cond-mat,physics:physics,physics:quant-ph].

[7] Edwin Pednault *et al.*, *Pareto-efficient quantum circuit simulation using tensor contraction deferral*, 2020. DOI: 10.48550/arXiv.1710.05867. arXiv: 1710.05867[quant-ph].

[8] Igor L. Markov and Yaoyun Shi, "Simulating quantum computation by contracting tensor networks," *SIAM J. Comput.*, vol. 38, no. 3, pp. 963–981, 2008. DOI: 10.1137/050644756. arXiv: quant-ph/0511069.

[9] Lukas Burgholzer, Alexander Ploier, and Robert Wille, "Exploiting arbitrary paths for the simulation of quantum circuits with decision diagrams," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, ISSN: 1558-1101, 2022, pp. 64–67. DOI: 10.23919/DATE54114.2022.9774631.

[10] Lukas Burgholzer, Hartwig Bauer, and Robert Wille, "Hybrid schrödinger-feynman simulation of quantum circuits with decision diagrams," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2021, pp. 199–206. DOI: 10.1109/QCE52317.2021.00037.

[11] Robert Wille, Stefan Hillmich, and Lukas Burgholzer, "Tools for quantum computing based on decision diagrams," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–17, 2022. DOI: 10.1145/3491246. arXiv: 2108.07027[quant-ph].

[12] Lukas Burgholzer, Alexander Ploier, and Robert Wille, *Tensor networks or decision diagrams? guidelines for classical quantum circuit simulation*, 2023. DOI: 10.48550/arXiv.2302.06616. arXiv: 2302.06616[quant-ph].

[13] Thomas Grurl *et al.*, "Arrays vs. decision diagrams: A case study on quantum circuit simulators," in *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)*, 2020, pp. 176–181. DOI: 10.1109/ISMVL49045.2020.000-9.

[14] Alwin Zulehner and Robert Wille, *Advanced simulation of quantum computations*, 2018. arXiv: 1707.00865 [quant-ph].

[15] Alwin Zulehner and Robert Wille, "Matrix-vector vs. matrix-matrix multiplication: Potential in dd-based simulation of quantum computations," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 90–95. DOI: 10.23919/DATE.2019.8714836.

[16] Holger F. Hofmann, "How to simulate a universal quantum computer using negative probabilities," *J. Phys. A: Math. Theor.*, vol. 42, no. 27, p. 275 304, 2009. DOI: 10.1088/1751-8113/42/27/275304.

[17] Tianyi Peng *et al.*, "Simulating large quantum circuits on a small quantum computer," *Phys. Rev. Lett.*, vol. 125, no. 15, p. 150 504, 2020. DOI: 10.1103/PhysRevLett.125.150504. arXiv: 1904.00102[quant-ph].

[18] Kosuke Mitarai and Keisuke Fujii, "Overhead for simulating a non-local channel with local channels by quasiprobability sampling," *Quantum*, vol. 5, p. 388, 2021. DOI: 10.22331/q-2021-01-28-388.

[19] Christophe Piveteau and David Sutter, "Circuit knitting with classical communication," *IEEE Trans. Inform. Theory*, vol. 70, no. 4, pp. 2734–2745, 2024. DOI: 10.1109/TIT.2023.3310797. arXiv: 2205.00016[quant-ph].

[20] Christian Ufrecht *et al.*, "Cutting multi-control quantum gates with ZX calculus," *Quantum*, vol. 7, p. 1147, 2023. DOI: 10.22331/q-2023-10-23-1147.

[21] Christian Ufrecht *et al.*, "Optimal joint cutting of two-qubit rotation gates," *Phys. Rev. A*, vol. 109, no. 5, p. 052 440, 2024. DOI: 10.1103/PhysRevA.109.052440.

[22] Lukas Schmitt, Christophe Piveteau, and David Sutter, *Cutting circuits with multiple two-qubit unitaries*, 2024. DOI: 10.48550/arXiv.2312.11638. arXiv: 2312.11638[quant-ph].

[23] Marvin Bechtold *et al.*, "Investigating the effect of circuit cutting in QAOA for the MaxCut problem on NISQ devices," *Quantum Sci. Technol.*, vol. 8, no. 4, p. 045 022, 2023, Publisher: IOP Publishing. DOI: 10.1088/2058-9565/acf59c.

[24] Hiroyuki Harada, Kaito Wada, and Naoki Yamamoto, *Doubly optimal parallel wire cutting without ancilla qubits*, 2023. DOI: 10.48550/arXiv.2303.07340. arXiv: 2303.07340[quant-ph].

[25] Angus Lowe *et al.*, "Fast quantum circuit cutting with randomized measurements," *Quantum*, vol. 7, p. 934, 2023. DOI: 10.22331/q-2023-03-02-934.

[26] Marvin Bechtold *et al.*, *Joint wire cutting with non-maximally entangled states*, 2024. DOI: 10.48550/arXiv.2406.13315. arXiv: 2406.13315[quant-ph].

[27] Michael A. Nielsen *et al.*, "Quantum dynamics as a physical resource," *pra*, vol. 67, no. 5, 052301, p. 052 301, 2003. DOI: 10.1103/PhysRevA.67.052301. arXiv: quant-ph/0208077 [quant-ph].

[28] Quantum AI team and collaborators, *Qsim*, 2020. DOI: 10.5281/zenodo.4023103. [Online]. Available: https://doi.org/10.5281/zenodo.4023103.

[29] Frank Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019, Publisher: Nature Publishing Group. DOI: 10.1038/s41586-019-1666-5.

[30] Peter L. Ivănescu, "Some network flow problems solved with pseudo-boolean programming," *Operations Research*, vol. 13, no. 3, pp. 388–399, 1965. DOI: 10.1287/opre.13.3.388. eprint: https://doi.org/10.1287/opre.13.3.388.

[31] F. Barahona, M. Jünger, and G. Reinelt, "Experiments in quadratic 0–1 programming," *Mathematical Programming*, vol. 44, no. 1–3, pp. 127–137, 1989. DOI: 10.1007/bf01587084.

[32] Caterina De Simone, "The cut polytope and the boolean quadric polytope," *Discrete Mathematics*, vol. 79, no. 1, pp. 71–75, 1990. DOI: 10.1016/0012-365x(90)90056-n.

[33] Michael Jünger and Sven Mallach, "Exact facetial odd-cycle separation for maximum cut and binary quadratic optimization," *INFORMS Journal on Computing*, 2021. DOI: 10.1287/ijoc.2020.1008.

[34] Laura S. Herzog *et al.*, *Improving quantum and classical decomposition methods for vehicle routing*, 2024. DOI: 10.48550/arXiv.2404.05551. arXiv: 2404.05551[quant-ph].

[35] Jonas Richter, *Simulating the dynamics of large many-body quantum systems with schr\"odinger-feynman techniques*, 2024. DOI: 10.48550/arXiv.2403.19864. arXiv: 2403.19864[cond-mat,physics:quant-ph].

[36] Sergio Boixo and Charles Neill. "The question of quantum supremacy." https://github.com/sboixo/GRCS. ().

[37] Mikhail Smelyanskiy, Nicolas P. D. Sawaya, and Alán Aspuru-Guzik, "qHiPSTER: The Quantum High Performance Software Testing Environment," *arXiv e-prints*, arXiv:1601.07195, arXiv:1601.07195, 2016. DOI: 10.48550/arXiv.1601.07195. arXiv: 1601.07195 [quant-ph].

[38] Thomas Häner and Damian S. Steiger, "0.5 Petabyte Simulation of a 45-Qubit Quantum Circuit," *arXiv e-prints*, arXiv:1704.01127, arXiv:1704.01127, 2017. DOI: 10.48550/arXiv.1704.01127. arXiv: 1704.01127 [quant-ph].

RAP-Track: Efficient Control Flow Attestation via Parallel Tracking in Commodity MCUs

Antonio Joia Neto
University of Zurich
ajneto@ifi.uzh.ch

Adam Caulfield
University of Waterloo
acaulfie@uwaterloo.ca

Ivan De Oliveira Nunes
University of Zurich
nunes@ifi.uzh.ch

***Abstract*—Control Flow Attestation (CFA) has emerged as an important security service to enable remote verification of control flow paths in safety-critical embedded systems. However, current CFA for commodity devices suffers performance penalties due to code instrumentation and frequent context switches required to securely log control flow paths at runtime. Our work introduces RAP-Track, a technique leveraging commodity hardware extensions, namely Micro Trace Buffer and Data Watchpoint and Trace Unit, to track control flow paths in parallel with the execution of the attested program, thus avoiding aforementioned overheads present in state-of-the-art CFA. Our evaluation (based on an open-source prototype of RAP-Track) demonstrates substantial performance gains, enhancing practicality and security of CFA.**

I. Introduction

Embedded systems play vital roles in interfacing the physical and digital worlds. They are often deployed remotely for on-demand sensing and/or actuation in critical operations. In these settings, low-power micro-controller units (MCUs) are preferred for their energy efficiency. MCUs – typically having low CPU frequencies with minimal resources – lack the robust security features found in fully-fledged computers, such as memory management units (MMUs), virtual memory, and inter-process isolation. As a result, they become prime targets for attacks [1].

In this landscape, Remote Attestation (RA) [2] has become a popular method to remotely verify the integrity of MCUs. In RA, a Verifier (Vrf) wants to ascertain the current software state of a remotely deployed Prover MCU (Prv). This is realized as a challenge-response protocol in which Vrf sends a challenge (in the form of a cryptographic nonce) to Prv. Upon receiving the challenge, a Root of Trust (RoT) in Prv is responsible for computing an authenticated integrity-ensuring function (e.g., a MAC or digital signature) on the received challenge and Prv's program memory. Based on the result, Vrf can determine whether Prv's code has been illegally modified.

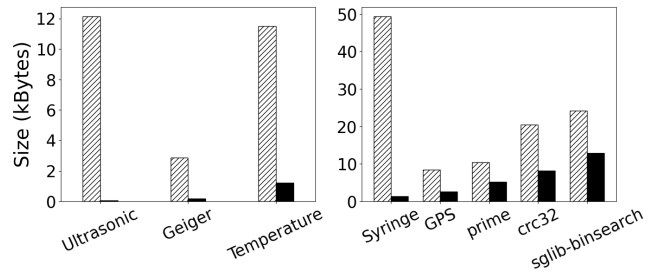
By itself, RA does not detect runtime attacks (e.g., Return/Jump-Oriented Programming [3], [4]) that corrupt software execution without modifying code. Control Flow Integrity (CFI) [5], [6], [7], [8], [9], [10], [11] can, to some extent, detect runtime attacks locally on Prv and abort execution. However, it does not produce attestable evidence of the unintended control flow paths that led to the exploit [12]. Alternatively, Control Flow Attestation (CFA) generates authenticated evidence of Prv's runtime behavior, granting remote visibility to runtime attacks. CFA augments RA by including Prv's execution control flow path (for a given Vrf-defined operation/function of interest) as part of the attestation evidence received by Vrf. To accomplish this, the RoT within Prv saves all control flow transfer destinations (e.g., due to jump, return, and call instructions) during an operation's execution in a control flow log ($\mathsf{CF}_{\mathsf{Log}}$). Then, $\mathsf{CF}_{\mathsf{Log}}$ is signed along with the received challenge and Prv's program memory and sent to Vrf for assessment.

Current CFA RoTs build $\mathsf{CF}_{\mathsf{Log}}$ with either: (1) customized hardware that interfaces with the CPU to detect/log control flow instructions in parallel to their execution [13], [14], [15], [16]; or (2) instrumentation of the attested application with calls to Trusted Execution Environment (TEE)-protected software that tracks the control flow path before returning to the application [17], [18], [19], [20], [21]. Both approaches involve trade-offs. Mechanisms relying on custom hardware require new MCU chip fabrication, making them unsuitable for immediate deployment in commercial devices. On the other hand, CFA mechanisms that use TEEs can be readily deployed on current devices. However, associated code instrumentation leads to significant overheads due to added instructions and frequent context switches into TEE-protected mode for control flow logging. Only one prior technique [22] leverages a commodity hardware feature – Intel Processor Trace (PT) – for CFA, thus reducing code instrumentation. However, this feature is only available in high-end Intel CPUs, making the approach infeasible in MCUs.

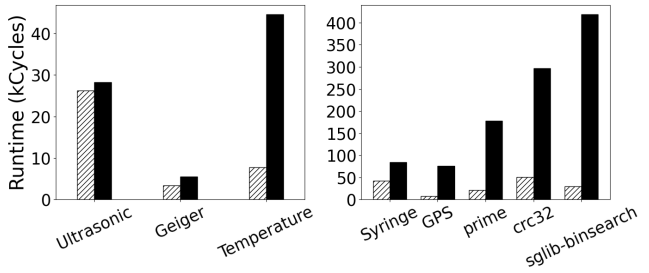
In terms of MCU-specific hardware features, ARM has introduced the Micro-Trace Buffer (MTB), enabling real-time control flow tracing of a program's execution [23]. Additionally, the Data Watchpoint and Trace (DWT) extension allows for monitoring of the processor's operations based on Program Counter (PC) values. Naturally, prior work has leveraged MTB for Control Flow Integrity (CFI) [5], [6] without instrumentation. Yet, the use of MTB for CFA introduces unique challenges that remain to be addressed. Unfortunately, MTB logs all control flow transfers, including (a very large number of) branches to fixed addresses (a.k.a. deterministic branches) that are unnecessary for CFA, given their fixed behavior. Since CFI processes this data locally on Prv, this overhead (while substantial) may be acceptable if Prv's execution can be periodically interrupted to examine and discard the generated control flow data. For CFA, since the entirety of $\mathsf{CF}_{\mathsf{Log}}$ is transmitted to Vrf, the latter creates a significant bottleneck.

In sum, despite reducing runtime overhead due to context switches and instrumentation, the *naive* usage of MTB to log transfers would generate much larger $\mathsf{CF}_{\mathsf{Log}}$-s than state-of-the-art TEE-based CFA. As a consequence, more memory to store $\mathsf{CF}_{\mathsf{Log}}$ would be demanded (on already limited MCUs), or constant interruptions to clear unnecessary $\mathsf{CF}_{\mathsf{Log}}$ entries would add delays to application runtime. Figure 1 illustrates this point, showing memory and timing demands of *naive* MTB-based logging in comparison to a state-of-the-art instrumentation-based CFA [24] on various open-source embedded applications: an ultrasonic ranger [25], a Geiger sensor [26], a Syringe Pump [27], a Temperature sensor [28], a GPS [29], as well as test applications from the BEEBs benchmark [30]. As seen in Figure 1(a), $\mathsf{CF}_{\mathsf{Log}}$ sizes generated by a *naive* MTB-based approach are significantly larger (1.9 to 217$\times$ larger on tested applications). On the other hand, instrumentation-based CFA adds significant runtime overheads (e.g., 1.1 to 14.1$\times$ increase on tested applications), as shown in Figure 1(b).

In this work, we propose an approach to combine MTB and DWT hardware extensions with an optimal strategy for linking of attested code sections. With this approach, we are able to reduce both $\mathsf{CF}_{\mathsf{Log}}$



(a) Comparison of $\mathsf{CF}_{\mathsf{Log}}$ size



(b) Comparison of Runtime Overhead



Fig. 1: Comparison $\mathsf{CF}_{\mathsf{Log}}$ sizes and runtime overhead of *naive* MTB- and Instrumentation-Based CFA

storage/transmission costs (due to *naive* MTB-based logging) and runtime overheads (due to code instrumentation and frequent context switches required by prior work).

Contribution. We propose RAP-Track: a method for Runtime Attestation via Parallel Tracking in commodity MCUs. RAP-Track leverages commodity hardware features of ARM MCUs – namely, MTB, DWT, and TrustZone (TZ) – in combination with strategic linking of attested code sections. DWT is used to define memory regions that control MTB activation and deactivation. RAP-Track offline static analysis module links code sections so that only branches that are essential for complete control flow reconstruction by Vrf are logged while also obviating context switches into TEE-protected mode during execution. We implement and evaluate RAP-Track on an ARM Cortex-M33 MCU using the V2M-MPS2-0318C prototyping system [31]. RAP-Track prototype is publicly available at [32].

II. Background & Related Work

A. Armv8-M Trustzone

TZ is a security extension for ARMv8 Cortex-M MCUs that creates a TEE by partitioning CPU resources into two separate domains: "Secure" and "Non-Secure" worlds. Each domain is configured with its own isolated memory and peripherals [33]. When the CPU is in a Non-Secure state, it can access only Non-Secure resources. In the Secure state, it can access both Secure and Non-Secure resources but is restricted to executing only Secure code. In the absence of virtual memory or MMUs, Cortex-M uses a Memory Protection Unit (MPU) for physical memory access control. With TZ, the MPU splits into Secure (S-MPU) and Non-Secure (NS-MPU) segments, with S-MPU access restricted to the Secure World and NS-MPU accessible to both worlds.

B. ARM Hardware Tracing Units

1) MicroTrace Buffer: The MTB [23] is a hardware extension introduced in ARM Cortex-M processors to enable lightweight, real-time control flow tracing of program execution. The MTB operates by recording every non-sequential memory address executed while it is activated. Unlike software-based instrumentation, which introduces significant performance overhead, the MTB operates with minimal impact on the CPU's performance.

The MTB uses a small, circular buffer located in the system's memory to store the trace data. As the program executes, the MTB captures branch instructions and writes the source address and destination address of the instruction into this buffer. In RAP-Track, this buffer contains $\mathsf{CF}_{\mathsf{Log}}$. Once the buffer is full, it wraps around and overwrites the oldest data, ensuring that the most recent execution flow is always available for analysis. It is possible to configure a Watermark (using MTB_FLOW register) to set a limit to the buffer that will generate a debug exception when the buffer reaches the Watermark. RAP-Track leverages this feature to generate partial reports (discussed further in Sec. IV-E).

There are two primary methods to activate MTB tracing. The first method involves directly setting the 'TSTARTEN' bit in the 'MTB_MASTER' register, which enables the recording of all non-sequential branches from that point forward. Alternatively, the 'MTB_TSTART' and 'MTB_TSTOP' registers can be used to start and stop the MTB tracing based on signals from the Data Watchpoint and Trace comparators.

2) Data Watchpoint and Trace: The DWT unit in the ARM Cortex-M33 processor [23] is a hardware extension that provides a way to monitor processor operations to specific memory ranges (such as reads, writes, and executions). It includes up to four configurable comparators, each of which can configured to generate *watchpoint* events when a specific memory address is accessed [34]. The DWT is integrated with the MTB, allowing DWT comparators to activate and deactivate the MTB Tracing based on DWT events. For example, two DWT comparators can be set to monitor a specific address range. Then, when the software within this range executes, the DWT triggers an event to MTB_START, initiating control flow tracing. Another set of DWT comparators can be configured to trigger MTB_STOP, ending the trace when the PC exits the specified range. RAP-Track uses the DWT in this way to activate/deactivate the MTB (discussed further in Sec. IV-B).

C. Remote Attestation

Remote attestation (RA) is a security service that allows a Verifier (Vrf) to assess the integrity of the code of a remote device, known as the Prover (Prv). The RA protocol usually follows four steps:

(1) Vrf creates a unique Chal and sends an RA request to Prv.
(2) Prv receives the request and measures its code or internal state, generating the attestation proof.
(3) Prv produces a proof that is authenticated with a signature or message authentication code (MAC).
(4) Vrf receives the proof and verifies whether Prv has a valid internal state.

The signature/MAC operation attests to Vrf that Prv's RoT is the generator of the proof. For this purpose, a secret key must be securely stored and managed exclusively by the RoT, ensuring it remains protected from any untrusted software on Prv. As such, RA mechanisms usually depend on hardware support (whether a fully hardware-based mechanism using standalone cryptographic co-processors [35], [36], [37] or a hybrid scheme using partial hardware support [38], [39], [40], [41], [42], [43], [44]).

D. Control Flow Attestation

Control Flow Attestation (CFA) extends RA by also providing evidence that an application's instructions were executed in a correct

sequence. This capability provides detection of Return-Oriented Programming (ROP) [3] and Jump-Oriented Programming (JOP) [4] attacks that can manipulate existing return/jump instructions to achieve Turing-complete behavior without altering the code.

CFA is achieved by requiring the RoT in Prv to generate and measure the control flow path followed during the application's execution. This requires recording all control flow transfers (e.g, due to jumps, calls, or returns) into CF_{Log}. Current CFA approaches employ either hardware [16], [14], [13], [15], [45], [22], [46] or instrumentation-based methods [47], [48], [17], [18], [24], [19], [20], [21], [49], [50] to create CF_{Log}. Hardware-based methods introduce custom hardware modules that interface with the CPU to detect/log control flow transfers as they occur [16], [14], [13], [15]. While efficient, these methods are incompatible with commodity devices. In contrast, instrumentation-based methods use existing TEE support (such as ARM TZ) for secure logging of an instrumented application [17], [18], [24], [19], [20], [21], [49].

TEE-based methods usually instrument all the control flow instructions in the application code, adding instructions that call the TEE (via dedicated non-secure callable entry points) to record each branch destination into CF_{Log}-dedicated Secure World memory. Due to frequent context switches, these approaches typically add significant runtime overhead, as each branch requires a transition from the Non-Secure to the Secure World to record the event. To mitigate this overhead, different CFA methods operate at varying levels of control flow granularity, selectively logging only critical code sections to reduce CF_{Log} overhead [19], [20]. We refer to these selective methods as "lossy" CFA, where Vrf cannot reconstruct the entire control flow path. In contrast, "lossless" CFA enables Vrf to fully recover the application's control flow path. The advantage of lossless CFA is that by recovering the full control flow path in CF_{Log}, Vrf can validate the entire execution path and observe any unintended/maliciously induced transitions. Additionally, lossless evidence could potentially grant visibility to attacks that do not directly overwrite branch destinations [12]. Naturally, CF_{Log}-s generated by lossless control flow mechanisms are larger, which correlates to more context switches for appending CF_{Log} [17], [18], [20], [19], [21], [24], [49].

To avoid runtime costs of instrumentation, dedicated hardware tracing components can be used, when available. Papamartzivanos et al. [22] proposed a technique that leverages Intel PT to trace low-level mission-critical processes. However, since Intel PT monitors everything within a code region, applying it alone can lead to large CF_{Log}-s. Thus, the authors consider analysis within the context of one target function. Additionally, [22] cannot be directly applied to embedded systems since it relies on Intel PT, which is only available in high-end Intel processors. A similar strategy is deployed by LAHEL [45], which uses off-chip debug features of ARM Coresight System Trace Macrocell technology in combination with TrustZone to implement CFA. In this case, the Program Trace Macrocell (PTM) [51] debug component is used to monitor program's execution in parallel. LAHEL leverages the PTM to log all branches taken and records the count of all branches not taken. Consequently, LAHEL records repeated branch-taken addresses due to loops. Additionally, LAHEL relies on external debug features rather than on-chip tracing units, such as MTB and DWT.

To our knowledge, MTB and DWT have only been used as a building block for CFI schemes [6], [5]. However, CFI has a fundamentally different goal than CFA: CFI mechanisms aim to locally detect control flow violations and abort execution upon detection, whereas CFA aims to generate authenticated evidence of the control flow path that was followed, whether it is benign or malicious [12]. Given their distinct goals, schemes that used MTB/DWT for efficient CFI cannot be applied directly to achieve efficient CFA. For instance, CFI methods utilizing MTB/DWT add additional steps into the local verification process that first filter redundant information from the MTB. Since CFA performs no local verification, the communication cost of CF_{Log} when applying MTB would be incredibly costly, nullifying improvements on the application's runtime.

III. System and Adversary Model

Prv is a single-core MCU equipped with a TEE (e.g., TZ in RAP-Track) running software at "bare-metal". The attested application (APP) executes in the Non-Secure World. RAP-Track implementation is housed in the Secure World (which is trusted). Prv is equipped with MTB and DWT hardware extensions. MTB and DWT are trusted to correctly implement their specification.

Adv fully controls Prv's Non-Secure World and is able to alter its code (e.g., via code injection attacks) or launch control flow/code reuse attacks. Given TZ protections (discussed in Section II-A), Adv cannot access or tamper with code or data in the Secure World. Similarly, it cannot deactivate or circumvent TZ hardware-enforced access control rules and assurances. Physical and hardware-modifying attacks require orthogonal tamper resistance measures [52] and are thus out our scope.

In this work, our treatment of CFA assumes that Non-Secure World interrupts are disabled during the execution of APP. We treat interrupt enablement as an orthogonal issue and refer the reader to [18], [53], [54] for a discussion on security implications and approaches that could complement RAP-Track by supporting safe interruptability.

IV. RAP-Track Design

RAP-Track tailors MTB/DWT usage for efficient CFA. RAP-Track's use of MTB reduces the number of Secure World calls for updating CF_{Log}, while its static linking strategy prevents logging irrelevant information (e.g., deterministic or repeated branch destinations) into CF_{Log} by MTB. As a result, RAP-Track reduces both the runtime cost of instrumentation and the storage cost of MTB-generated CF_{Log}-s. As shown in Figure 2, RAP-Track high-level operation includes:

Offline Phase. Prior to code deployment, RAP-Track static analysis phase divides APP into two regions: the MTB Activation Region (MTBAR) and the MTB Deactivation Region (MTBDR) (see Section IV-B for details on each region). All deterministic control flow transfers of APP are placed in the MTBDR, while non-deterministic transfers (e.g., indirect jumps/calls, returns, and conditional branches) are moved to the MTBAR. Their previous locations in APP are replaced with direct branches (i.e., *trampolines*) targeting their new addresses in MTBAR (details in Section IV-C). When the MCU executes a trampoline instruction, it directly branches to the MTBAR region, where the original branch operation is executed and thus measured by the MTB. This approach ensures that only non-deterministic branch destinations are recorded to CF_{Log}.

Execution Phase. When Prv receives a request for CFA of APP, it initializes the CFA Engine (Section IV-A) to start the CFA process. In RAP-Track this includes configuring MTB to track APP 's control flow path. APP execution then starts in the Non-Secure World. During APP's execution, branches that were linked within MTBAR are logged to CF_{Log} by the MTB. Once APP execution concludes, an authenticated CFA report (as outlined in Sections II-C and II-D) is generated and sent to Vrf.

A. CFA *Engine*

To start CFA of an APP, the CFA Engine disables all the Non-Secure interrupts and configures the NS-MPU to make APP's binary

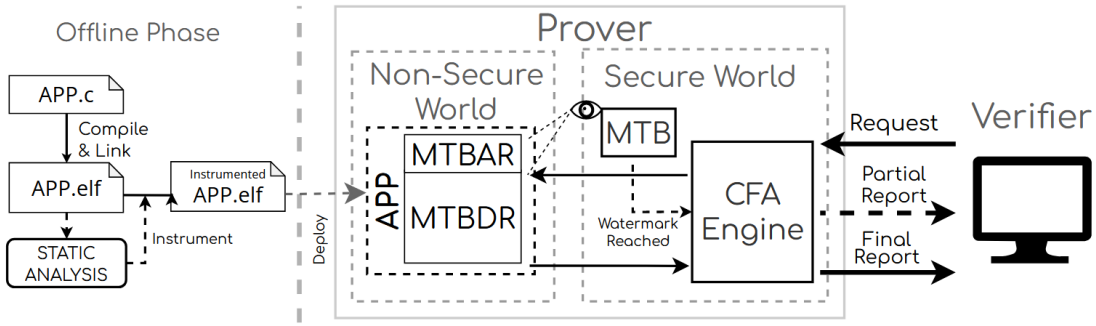


Fig. 2: System Model

non-writable. The NS-MPU is then locked to prevent modifications by the Non-Secure World, following prior CFA methods (e.g., [24]). Afterward, the CFA engine hashes all code associated with APP, producing H_{MEM}. Next, it configures the DWT and MTB so that MTB is enabled while executing within MTBAR and disabled while executing within MTBDR. After this setup, APP execution is called in the Non-Secure World. As a consequence of this configuration, for each section of APP code executed in MTBAR, the MTB logs the source and destination of all non-sequential branch operations to CF_{Log}. Once execution concludes, RAP-Track implementation in the Secure World signs a report containing Chal, H_{MEM}, and CF_{Log} using a private key stored within the Secure World and sends the report to Vrf.

B. DWT-based MTBAR and MTBDR Activation

RAP-Track static analysis divides APP code layout into MTBAR and MTBDR based on the instruction type, and inserts trampolines to connect them (discussed further in Section IV-C). All instructions with non-deterministic branch destinations are added into MTBAR, and everything else is added into MTBDR (i.e., non-branch instructions and deterministic branches). Based on this organization of instructions, RAP-Track must activate the MTB when executing the MTBAR region, and deactivate when executing the MTBDR region. Activation/deactivation is done via DWT as follows:

MTBAR: Two DWT comparators are used to define the MTBAR's boundaries: one for the base address and the other for the upper limit address. DWT then automatically activates the MTB when the address of the currently executing instruction (PC) is within these bounds. This is achieved by having the DWT set the 'MTB_TSTART' register when PC points to an address within MTBAR, thus activating the MTB. Based on this configuration, MTB does not record transitions from MTBDR into MTBAR.

MTBDR: Two additional DWT comparators are used in a similar fashion, except in this case they set 'MTB_TSTOP' register when PC is within the bounds of MTBDR. Based on this configuration, MTB does record transitions from MTBAR to MTBDR.

C. Branch Trampolines

This section explains how RAP-Track instruments different types of branch instructions to be able to track APP's control flow.

Statically Deterministic Branches. RAP-Track does not track static branches, such as jumps to constants and simple loops with fixed iteration counts because their destinations cannot be modified at runtime. Static branch instructions remain in MTBDR, as their targets are can be obtained statically by Vrf from APP's binary [19], [24] and need not be logged.

Non-Deterministic Branches includes branches caused by indirect jumps/calls, function returns, loops with variable iteration counts, and loops with internal branches, where the target address is not static/deterministic. Target addresses of these instructions must be recorded in CF_{Log}, as they are necessary for Vrf to reconstruct the entire control flow path. These instructions are moved from their original location in MTBDR to MTBAR. *Trampoline* instructions targeting the new addresses in MTBAR replace the original branch instructions in MTBDR. The *trampoline* type depends on the type of non-deterministic branch instruction.

Original Code	**Modified Code**

`(1) BLX Register`	`(1) BL MTBAR_ADDRESS`
	`...`
	`MTBAR_ADDRESS:`
	`BX Register`

Fig. 3: Trampoline for indirect call

1) Indirect Calls.: Indirect calls perform a call to a destination specified in a register. Figure 3 shows the trampoline placed by RAP-Track for indirect calls. Each indirect call is replaced with a direct call to a fixed address within the MTBAR (`MTBAR_ADDRESS` in Figure 3). Then, at `MTBAR_ADDRESS`, an indirect branch to the original register is placed.

Original Code	**Modified Code**

`POP {PC}`	`B MTBAR_POP_ADDR`
	`...`
	`MTBAR_POP_ADDR:`
	`POP {PC}`

Original Code	**Modified Code**

`LDR PC, [sp, #4]`	`B MTBAR_LDR_ADDR`
	`...`
	`MTBAR_LDR_ADDR:`
	`LDR PC, [sp, #4]`

Fig. 4: Trampolines for returns and indirect jumps

2) Returns and Indirect Jumps.: In ARMv8-M, when a call instruction is executed, the return address is stored in a reserved register called the Link Register (`LR`). The compiler pushes `LR` onto the stack only if the called function contains a nested function call; otherwise, the return is performed using a branch to `LR`. In this case, if no operations within the function modify `LR`, the return address remains unchanged throughout the function's execution, making it predictable. Therefore, RAP-Track does not monitor these cases and instead focuses on monitoring return operations only when the contents of `LR` have been pushed onto the stack.

Given this insight, monitored return instructions and indirect jumps (used for C-switch statements) have equivalent implementations as either: (I) a `POP` to the PC; (II) or a memory load `LDR` into PC.

RAP-Track inserts trampolines for both cases in the same way, as depicted in Figure 4. First, a direct branch to the MTBAR is inserted in place of the original return/indirect jump instruction. A direct branch to `MTBAR_POP_ADDR` (or `MTBAR_LDR_ADDR`) is used to replace a return/indirect jump via `POP` (or `LDR`) instruction. Once in the MTBAR, the original `POP` (or `LDR`) instruction is executed to complete the return/indirect jump.

3) Conditional Branches: In C, both if/else statements and any form of loop use conditional branch instructions. To optimize the information recorded by MTB to CF_{Log}, RAP-Track conditional branch trampoline varies in three cases:

(1) non-loop conditional branches
(2) loops conditional branches that jump "backward"
(3) loops conditional branches that jump "forward"

Original Code

```
    CMP R0,#0
(1) BEQ taken_address
```

Modified Code

```
    CMP R0,#0
(1) BEQ MTBAR_ADDRESS
    ...
MTBAR_ADDRESS:
    B taken_address
```

Fig. 5: Trampoline for non-loop conditional branches

3.1) Non-loop Conditional Branches. When a conditional branch is not part of a loop (e.g., used to implement an if/else statement), compilers often optimize the conditional branches of if/else statements by branching to the less likely path, leaving the more probable path as a fall-through, which helps avoid CPU pipeline stalls. Based on this insight, RAP-Track inserts a trampoline so that MTB only records the branch-taken address to minimize the size of CF_{Log}, as it is the less likely direction of the instruction (where the most likely path is implicitly represented by the absence of an entry in CF_{Log}). In the example of Figure 5, the branch-taken address of conditional branch instruction (`BEQ`) is replaced with a fixed address in MTBAR. Then in MTBAR, a direct branch is added that jumps to the original branch-taken address.

Original Code

```
loop_start:
    ...
    CMP R0,#0
(1) BEQ loop_start
```

Modified Code

```
loop_start:
    ...
    CMP R0,#0
(1) BEQ MTBAR_ADDRESS
    ...
MTBAR_ADDRESS:
    B loop_start
```

Fig. 6: Trampoline for backward loop conditional branches

3.2) Backward Loop Conditional Branches. In contrast to non-loop conditional branches, the more likely branch destination depends on whether the loop conditional branch is implemented with a "backward" or "forward" conditional branch. First, we consider a loop that is implemented with a backward conditional branch. We refer to "backward" conditional branches as those with a destination address that is smaller than the current instruction address. When implemented with a backward conditional branch, the branch destination indicates the loop's start, as depicted in Figure 6. RAP-Track should ensure CF_{Log} reflects all iterations of the loop to ensure evidence is lossless. Therefore, a trampoline similar to the non-loop condition case is added to log each branch-taken address.

3.3) Forward Conditional Branches. We refer to conditional branches where the destination address is greater than the current address as "forward" branches. Loops, in some cases, can be implemented with forward branches. When this occurs, the branch-taken address is the loop's exit, and the loop is continued through the branch-not-taken path. Therefore in order to record each loop iteration in CF_{Log}, RAP-Track inserts trampolines for forward conditional loop branches to log the not-taken address. This is depicted in Figure 7.

Original Code

```
    CMP R0,#0
    BEQ loop_end
    ADD R1, R2, R3
    ...
loop_end:
    ...
```

Modified Code

```
    CMP R0,#0
    BEQ loop_end
(1)-> B MTBAR_ADDRESS
br_not_taken_addr:
    ADD R1, R2, R3
    ...
loop_end:
    ...
MTBAR_ADDRESS:
    B br_not_taken_addr
```

Fig. 7: Trampoline for forward loop conditional branches

In the modified code, the compare and conditional branches are not modified since they implement the loop exit. Instead, RAP-Track inserts a direct branch directly after the conditional branch to the MTBAR to log the branch-not-taken address. This branch jumps to a fixed address in the MTBAR that trampolines back to the modified code at the original branch-not-taken address. In this example, it branches back to `br_not_taken_addr`, the location of the `ADD` instruction that previously followed the loop conditional branch.

Adding the loop trampolines depicted in Figures 6 and 7 is only required when there are non-deterministic branch instructions within the loop or when the loop condition is based on a variable.

D. Loop Optimization

Several CFA methods (e.g., [17], [19], [24], [15]) add loop optimizations to avoid redundant CF_{Log} entries. To reduce the impact of loops on CF_{Log}, RAP-Track adds minimal instrumentation to log the loop condition itself, similarly to prior work [49], [24]. RAP-Track adds this instrumentation when the loop comparison is made to a fixed constant value, the loop iterator is based on register-only operations (e.g., only arithmetic instructions rather than loads/stores), and when its internal branches are all deterministic. These "simple" loop characteristics are very common in MCU applications (e.g., to implement data operations over buffers or control loops that sense data for fixed periods). The instrumentation adds a call to the CFA Engine before the loop entry to log the loop condition in CF_{Log}.

E. Partial Reports

Prv might be equipped with limited memory to store CF_{Log} that fills in the middle of APP execution. To manage this, RAP-Track sets the 'MTB_FLOW' register to limit CF_{Log}'s size. When CF_{Log} size equals the watermark, an exception is triggered in the Secure World to generate and transmit a partial report to Vrf. Afterward, the head pointer of CF_{Log} is reset, and then APP resumes, allowing MTB to overwrite the same memory designated for storing CF_{Log}.

F. Security Analysis

To bypass RAP-Track, Adv must alter control flow in a way that goes undetected while producing valid proof accepted by the Vrf. One approach is modifying APP binary directly. However, CFA Engine hashes and locks APP 's memory: any changes trigger a memory fault, invalidating the report. Attempts to manipulate control flow without detection fail, as all indirect branches are logged to CF_{Log} by MTB, and CF_{Log} is built and stored within protected memory in the Secure World. Forging or replaying the report is also infeasible as long as a cryptographically secure signature (or MAC, in the symmetric setting) and fresh challenge are used for each CFA request. Finally, Adv cannot deactivate/misconfigure MTB or DWT because they are only configurable by the Secure World.

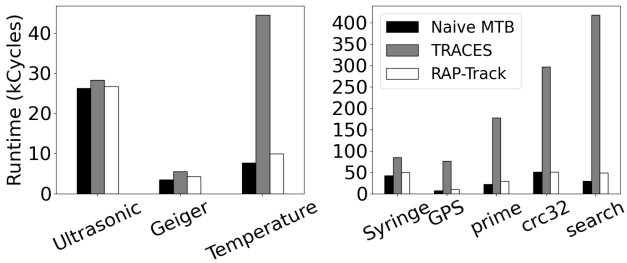


Fig. 8: Runtime Comparison

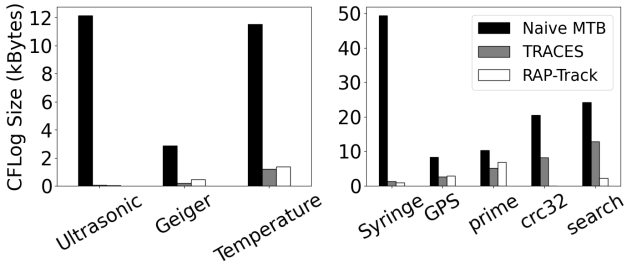


Fig. 9: $\mathsf{CF}_{\mathsf{Log}}$ Size Comparison

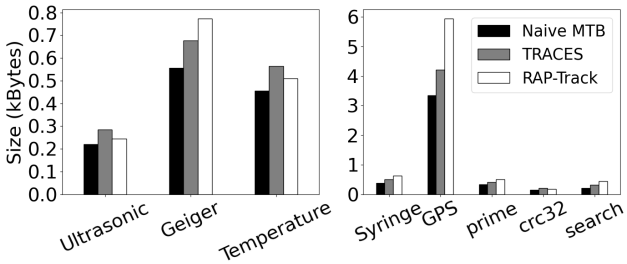


Fig. 10: Code size comparison

V. Implementation & Evaluation

We implemented RAP-Track proof-of-concept prototype (publicly available at [32]) on the V2M-MPS2-0318C prototyping system [31], which is equipped with an AN505 Arm Cortex-M33 FPGA image [55] based on the ARM Cortex-M33 (v8) architecture with support to TZ, MTB, and DWT extensions. RAP-Track Secure World code occupies 11 kilobytes (KB) in total with the CFA Engine occupying 2.8 KB. RAP-Track static analysis/linking code used in the Offline Phase is written in Python and operates directly on post-compiled binaries. We evaluate RAP-Track's memory and runtime overhead on real MCU applications (the same open-source applications listed in Section I). These applications are the same used in prior work, enabling direct comparison. RAP-Track performance is compared to TRACES [24]: a TEE-based CFA architecture implementing state-of-the-art $\mathsf{CF}_{\mathsf{Log}}$ optimizations. We note that, while other CFA architectures (e.g., [17], [19]) also implement optimizations similar to TRACES, TRACES open-source availability enables our comparison (whereas we are unable to compare to closed source CFA architectures). In addition to TRACES, as a baseline for runtime, we consider unmodified APP-s without any CFA-enabling instrumentation and associated overheads. For the baseline on $\mathsf{CF}_{\mathsf{Log}}$ size and code size, we use the *naive* MTB-based logging discussed in Section I.

A. Runtime Overhead

The runtime overhead of instrumentation-based CFA methods is primarily due to three factors: trampolines that invoke the CFA Engine API, context switches between the Non-Secure and Secure Worlds, and the logging algorithm (along with optimizations). In contrast, RAP-Track's overhead mainly comes from trampoline instructions for branching between MTBAR and MTBDR, with the only context switches required being to log loop conditions. The *naive* MTB approach incurs no overhead as it lacks any instrumentation. Figure 8 shows the CPU cycles required by each method in the sample APP-s. RAP-Track adds from 2% to 62% in CPU cycles over the *naive* MTB approach, whereas TRACES overhead ranges from 7% to 1309%. Differences across APP-s are due to the varying frequency of control flow transfers in their implementations.

B. $\mathsf{CF}_{\mathsf{Log}}$ Size

The size of $\mathsf{CF}_{\mathsf{Log}}$-s generated by each method is shown in Figure 9. $\mathsf{CF}_{\mathsf{Log}}$ sizes of RAP-Track and TRACES across APP-s vary depending on the optimizations and execution paths taken in each case. For instance, RAP-Track only records the branch taken in if-else statements, so the frequency of each path directly impacts $\mathsf{CF}_{\mathsf{Log}}$ size. Additionally, loop optimizations can substantially reduce $\mathsf{CF}_{\mathsf{Log}}$ size, as demonstrated in the results from the Ultrasonic Sensor and Syringe applications, highlighting that RAP-Track produces significantly smaller $\mathsf{CF}_{\mathsf{Log}}$-s than *naive* MTB. The differences in $\mathsf{CF}_{\mathsf{Log}}$ size between RAP-Track and TRACES depend on the specific application and optimizations used. It is also possible to implement instrumentation-based CFA that records the exact branches tracked by RAP-Track. Results on *prime* and *gps* applications (from the BEEBs benchmark [30]) show that RAP-Track achieves considerably better runtime while producing similarly sized $\mathsf{CF}_{\mathsf{Log}}$-s. We also note that in lossless CFA, Prv must pause APP to send partial reports when $\mathsf{CF}_{\mathsf{Log}}$ assigned memory reaches capacity. Thus, $\mathsf{CF}_{\mathsf{Log}}$ size directly impacts communication overhead/latency, often becoming the system's primary bottleneck (see [56]). For example, the ARM-M33 MTB used in our prototype has a 4KB limit, which would cause applications that use a *naive* MTB approach to frequently pause for partial report transmissions in most cases. In contrast, RAP-Track can fit all transfers within 4KB in most of the tested applications, only requiring one transmission to Vrf (once APP execution completes) and thus avoiding pauses.

C. Program Memory Overhead

Figure 10 presents the code size of each application, showcasing the impact of instructions introduced by RAP-Track trampolines and the TRACES instrumentation. In most APP-s, RAP-Track incurs slightly more code size overhead, with a significant variation based on the density of indirect and conditional branches. This additional overhead is primarily due to RAP-Track requiring extra instructions in conditional loops compared to TRACES. Additionally, 'nop' instructions were added in MTBAR trampolines to allow the MTB sufficient time to activate before logging a branch, as the MTB hardware activation is not immediate upon entering the MTBAR.

VI. Conclusion

This paper presents RAP-Track, a TEE-based CFA mechanism to minimize instrumentation and context switches in "off-the-shelf" commodity MCUs. RAP-Track static linking organizes all non-deterministic branches into a monitored region to optimize MTB-logging for CFA by leveraging PC-based hardware controls from the DWT extension. RAP-Track offers considerably more efficient runtime while maintaining a similar $\mathsf{CF}_{\mathsf{Log}}$ size as state-of-the-art methods. RAP-Track prototype is publicly available at [32].

Acknowledgments

We thank DAC anonymous reviewers for their helpful comments. Part of this work was conducted while the authors were at RIT.

REFERENCES

[1] M. N. Nafees, N. Saxena, A. Cardenas, S. Grijalva, and P. Burnap, "Smart grid cyber-physical situational awareness of complex operational technology attacks: A review," *Computing Surveys*, 2023.

[2] I. D. O. Nunes, S. Jakkamsetti, N. Rattanavipanon, and G. Tsudik, "Towards remotely verifiable software integrity in resource-constrained iot devices," *IEEE Communications Magazine*, 2024.

[3] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications," *TISSEC*, 2012.

[4] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jump-oriented programming: a new class of code-reuse attack," in *AsiaCCS*, 2011.

[5] X. Tan and Z. Zhao, "Sherloc: Secure and holistic control-flow violation detection on embedded systems," in *CCS*. ACM, 2023.

[6] Y. Wang, C. L. Mack, X. Tan, N. Zhang, Z. Zhao, S. Baruah, and B. C. Ward, "Insectacide: Debugger-based holistic asynchronous cfi for embedded system," in *RTAS*. IEEE, 2024.

[7] X. Ge, W. Cui, and T. Jaeger, "Griffin: Guarding control flows using intel processor trace," *SIGPLAN Notices*, 2017.

[8] ARM, "BTI," https://developer.arm.com/documentation/ddi0602/2021-12/Base-Instructions/BTI--Branch-Target-Identification-, 2020, [Online; accessed 13-February-2023].

[9] ——, "Return Address Signing using ARM Pointer Authentication," https://gcc.gnu.org/legacy-ml/gcc-patches/2018-11/msg00104.html, 2018, [Online; accessed 13-February-2023].

[10] Microsoft, "Control Flow Guard for platform security," https://learn.microsoft.com/en-us/windows/win32/secbp/control-flow-guard, 2022, [Online; accessed 13-February-2023].

[11] N. S. Almakhdhub, A. A. Clements, S. Bagchi, and M. Payer, "μRAI: Securing Embedded Systems with Return Address Integrity," in *NDSS*, 2020.

[12] M. Ammar, A. Caulfield, and I. D. O. Nunes, "Sok: Integrity, attestation, and auditing of program execution," in *S&P*. IEEE, 2025.

[13] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "Lo-fat: Low-overhead control flow attestation in hardware," in *DAC*, 2017.

[14] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A.-R. Sadeghi, "Atrium: Runtime attestation resilient under memory attacks," in *ICCAD*. IEEE, 2017.

[15] A. Caulfield, N. Rattanavipanon, and I. D. O. Nunes, "ACFA: Secure runtime auditing & guaranteed device healing via active control flow attestation," in *USENIX Security Symposium*, 2023.

[16] G. Dessouky, T. Abera, A. Ibrahim, and A.-R. Sadeghi, "Litehax: lightweight hardware-assisted attestation of program execution," in *ICCAD*. IEEE, 2018.

[17] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-flat: control-flow attestation for embedded systems software," in *ACM CCS*, 2016.

[18] A. J. Neto and I. D. O. Nunes, "Isc-flat: On the conflict between control flow attestation and real-time operations," in *RTAS*. IEEE, 2023.

[19] Z. Sun, B. Feng, L. Lu, and S. Jha, "Oat: Attesting operation integrity of embedded devices," in *S&P*. IEEE, 2020.

[20] J. Wang, Y. Wang, A. Li, Y. Xiao, R. Zhang, W. Lou, Y. T. Hou, and N. Zhang, "ARI: Attestation of real-time mission execution integrity," in *USENIX Security Symposium*, 2023.

[21] N. Yadav and V. Ganapathy, "Whole-program control-flow path attestation," in *ACM CCS*, 2023.

[22] D. Papamartzivanos, S. A. Menesidou, P. Gouvas, and T. Giannetsos, "Towards efficient control-flow attestation with software-assisted multi-level execution tracing," in *MeditCom*. IEEE, 2021.

[23] A. Developer, *Arm® CoreSight™ MTB-M33 Technical Reference Manual*, 2024.

[24] A. Caulfield *et al.*, "Traces: Tee-based runtime auditing for commodity embedded systems," 2024.

[25] Seeed-Studio, "Ultrasonic Ranger," Jun. 2015. [Online]. Available: https://github.com/Seeed-Studio/LaunchPad_Kit/tree/master/Grove_Modules/ultrasonic_ranger

[26] Y. Tournade, "ArduinoPocketGeiger Github Repository," https://github.com/MonsieurV/ArduinoPocketGeiger, 2020.

[27] T. Walker, "OpenSyringePump," Apr. 2022. [Online]. Available: https://github.com/manimino/OpenSyringePump

[28] Seeed-Studio, "Temperature Sensor," https://github.com/Seeed-Studio/LaunchPad_Kit/tree/master/Grove_Modules/temp_humi_sensor, Jun. 2015.

[29] M. Hart, "Tinygps++," http://arduiniana.org/libraries/tinygpsplus/, 2014.

[30] J. Pallister, S. Hollis, and J. Bennett, "Beebs: Open benchmarks for energy measurements on embedded platforms," *arXiv preprint arXiv:1308.5174*, 2013.

[31] A. Developer, *Versatile Express V2M-MPS2 motherboard*, 2024. [Online]. Available: https://www.keil.com/boards2/arm/v2m_mps2_iot_/#/eula-container

[32] A. J. Neto, A. Caulfield, and I. D. O. Nunes, "RAP-Track prototype repository," https://github.com/SPINS-RG/RAP-Track, 2025.

[33] A. Ltd, "Trustzone technology for armv8-m architecture version 2.1," https://developer.arm.com/documentation/100690/0201/, 2019.

[34] A. Limited, "Dwt-functional-descriptionl," 2024. [Online]. Available: https://developer.arm.com/documentation/ddi0439/b/Data-Watchpoint-and-Trace-Unit/DWT-functional-description?lang=en

[35] X. Kovah, C. Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth, "New results for timing-based attestation," in *S&P*. IEEE, 2012.

[36] D. Schellekens, B. Wyseur, and B. Preneel, "Remote attestation on legacy operating systems with trusted platform modules," *Science of Computer Programming*, 2008.

[37] J. Noorman, J. V. Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling, "Sancus 2.0: A low-cost security architecture for iot devices," *TOPS*, 2017.

[38] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified Hardware/Software Co-Design for remote attestation," in *USENIX Security Symposium*, 2019.

[39] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "SMART: Secure and minimal architecture for (establishing dynamic) root of trust," in *NDSS*, 2012.

[40] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: A security architecture for tiny embedded devices," in *EuroSys*, 2014.

[41] K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "HYDRA: hybrid design for remote attestation (using a formally verified microkernel)," in *WiSec*. ACM, 2017.

[42] M. Ammar *et al.*, "Delegated attestation: scalable remote attestation of commodity cps by blending proofs of execution with software attestation," in *ACM WiSec*, 2021, pp. 37–47.

[43] I. De Oliveira Nunes *et al.*, "Casu: Compromise avoidance via secure update for low-end embedded systems," in *IEEE/ACM ICCAD*, 2022, pp. 1–9.

[44] ——, "Parsel: Towards a verified root-of-trust over sel4," in *IEEE/ACM ICCAD)*. IEEE, 2023.

[45] O. Arias, D. Sullivan, H. Shan, and Y. Jin, "Lahel: Lightweight attestation hardening embedded devices using macrocells," in *HOST*. IEEE, 2020.

[46] M. Geden and K. Rasmussen, "Hardware-assisted remote runtime attestation for critical embedded systems," in *PST*. IEEE, 2019.

[47] Y. Zhang *et al.*, "ReCFA: resilient control-flow attestation," in *ACSAC*, 2021.

[48] F. Toffalini, E. Losiouk, A. Biondo, J. Zhou, and M. Conti, "ScaRR: Scalable runtime remote attestation for complex systems," in *RAID*, 2019.

[49] I. D. O. Nunes, S. Jakkamsetti, and G. Tsudik, "Tiny-CFA: Minimalistic control-flow attestation using verified proofs of execution," in *DATE*. IEEE, 2021.

[50] ——, "Dialed: Data integrity attestation for low-end embedded devices," in *DAC*. IEEE, 2021.

[51] A. Limited, "Coresight system trace macrocell technical reference manual," 2024. [Online]. Available: https://developer.arm.com/documentation/ddi0444/latest/

[52] J. Obermaier *et al.*, "The past, present, and future of physical security enclosures: From battery-backed monitoring to puf-based inherent security and beyond," *Journal of Hardware and Systems Security*, vol. 2, 2018.

[53] A. J. Neto *et al.*, "Pearts: Provable execution in real-time embedded systems," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 47–47.

[54] A. Caulfield *et al.*, "Asap: reconciling asynchronous real-time operations and proofs of execution in simple embedded systems," in *DAC*, 2022, pp. 721–726.

[55] A. Developer, *Cortex™-M33 with IoT kit FPGA for MPS2+ Version 2.0 (AN505)*, 2024. [Online]. Available: https://developer.arm.com/downloads/view/AN505?sortBy=availableBy&revision=r0p0-00rel0

[56] A. Caulfield *et al.*, "Speccfa: Enhancing control flow attestation/auditing via application-aware sub-path speculation," *ACSAC*, 2024.

Insights from Rights and Wrongs: A Large Language Model for Solving Assertion Failures in RTL Design

Jie Zhou[1,2], Youshu Ji[2], Ning Wang[3], Yuchen Hu[1,2], Xinyao Jiao[1,2], Bingkun Yao[3], Xinwei Fang[4*], Shuai Zhao[5*], Nan Guan[3], Zhe Jiang[1,2*]
[1]School of Integrated Circuits, Southeast University, China
[2]National Center of Technology Innovation for EDA, China
[3]Department of Computer Science, City University of Hong Kong, Hong Kong
[4]Department of Computer Science, University of York, UK
[5]Department of Computer Science, Sun Yat-sen University, China

***Abstract*—SystemVerilog Assertions (SVAs) are essential for verifying Register Transfer Level (RTL) designs, as they can be embedded into key functional paths to detect unintended behaviours. During simulation, assertion failures occur when the design's behaviour deviates from expectations. Solving these failures, i.e., identifying and fixing the issues causing the deviation, requires analysing complex logical and timing relationships between multiple signals. This process heavily relies on human expertise, and there is currently no automatic tool available to assist with it. Here, we present AssertSolver, an open-source Large Language Model (LLM) specifically designed for solving assertion failures. By leveraging synthetic training data and learning from error responses to challenging cases, AssertSolver achieves a bug-fixing pass@1 metric of 88.54% on our testbench, significantly outperforming OpenAI's o1-preview by up to 11.97%. We release our model and testbench for public access to encourage further research: https://github.com/SEU-ACAL/reproduce-AssertSolver-DAC-25.**

I. Introduction

Functional verification is a crucial step in the modern Electronic Design Automation (EDA) process, ensuring that designs meet their specifications and perform correctly as intended [1], thereby mitigating the costly risks associated with silicon failures [2], [3]. SystemVerilog Assertions (SVAs), as one of the key methods in functional verification [4], capture potential errors in Register Transfer Level (RTL) designs by defining logical conditions and timing requirements. Unlike stimulus-based verification methods (e.g., using testbenches), SVAs not only facilitate stimulus-triggered checks but also enable formal verification [5]. Formal verification, which mathematically proves the consistency of the design under test (DUT) with the behaviour specified by the SVAs, provides higher functional coverage and addresses boundary conditions that stimulus-based methods may often overlook.

Despite the significant advantages of SVAs and progress in automating their generation [6]–[11], automatically solving assertion failures remains challenging. Assertion failures occur when the design exhibits unexpected behaviour during simulation. Effectively identifying and fixing these failures still relies heavily on human expertise to analyse intricate logical dependencies and timing relationships between multiple signals, which is time-consuming and labour-intensive. Fig. 1 illustrates that verification engineers must have an in-depth understanding of the design's functional intent and carefully deduce the causes of complex assertion failures.

Large Language Models (LLMs) have demonstrated significant capabilities in navigating through complex ideas, automating key steps in hardware design. They have improved a range of tasks, from code generation [12]–[16] to design verification [17]–[19],

*Corresponding author:
Zhe Jiang. Email: zhejiang.uk@gmail.com.
Xinwei Fang. Email: xinwei.fang@york.ac.uk.
Shuai Zhao. Email: zhaosh56@mail.sysu.edu.cn

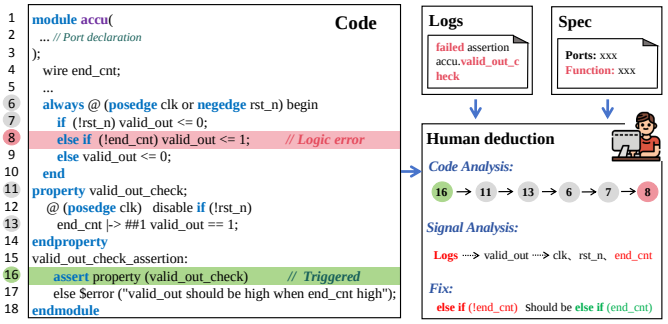


Fig. 1: The process of solving assertion failures in RTL verification. Engineers analyse the design specification and code signals to identify the root causes of assertion failures and implement fixes.

offering potential for solving assertion failures. However, while state-of-the-art (SOTA) LLMs such as OpenAI's GPT-4 and o1-preview demonstrate competence in such task, their performance is often subject to unnoticed changes, and they lack the ability to be retrained or fine-tuned by users to incorporate new data or information.

To address this limitation, we introduce an open-source LLM, AssertSolver, specifically designed for automatically solving assertion failures. As shown in Fig. 2, AssertSolver benefits from an innovative data augmentation method that enhances the representation of various assertion failure scenarios in the training dataset, which were previously underrepresented. AssertSolver is retrained and fine-tuned from the Deepseek-Coder-6.7b [20], using the dataset that incorporates not only significantly enhanced examples but also error responses to challenging cases. By learning from these "right" and "wrong" examples, AssertSolver demonstrates up to a 11.97% improvement in solving assertion failures compared to the OpenAI's o1-preview.

The main contributions of this paper are:

- Development of AssertSolver, an open-source domain-specific LLM fine-tuned for solving assertion failures, which we have made publicly available for early adoption;
- Implementation of a data augmentation method that automatically enrich scarce training dataset for assertion failures;
- Adoption of a new training strategy that improves learning from error responses to challenging cases;
- Publication of openly accessible benchmark for solving assertion failures, featuring over 900 instances across various bug types.

The rest of the paper is organised as follows: Section II introduces the data augmentation approach and the preparation of the training dataset. Section III provides a detailed description of our training strategies. Section IV outlines the research questions and the experimental setup. Section V presents a comprehensive evaluation to answer the research questions. Section VI concludes the paper.

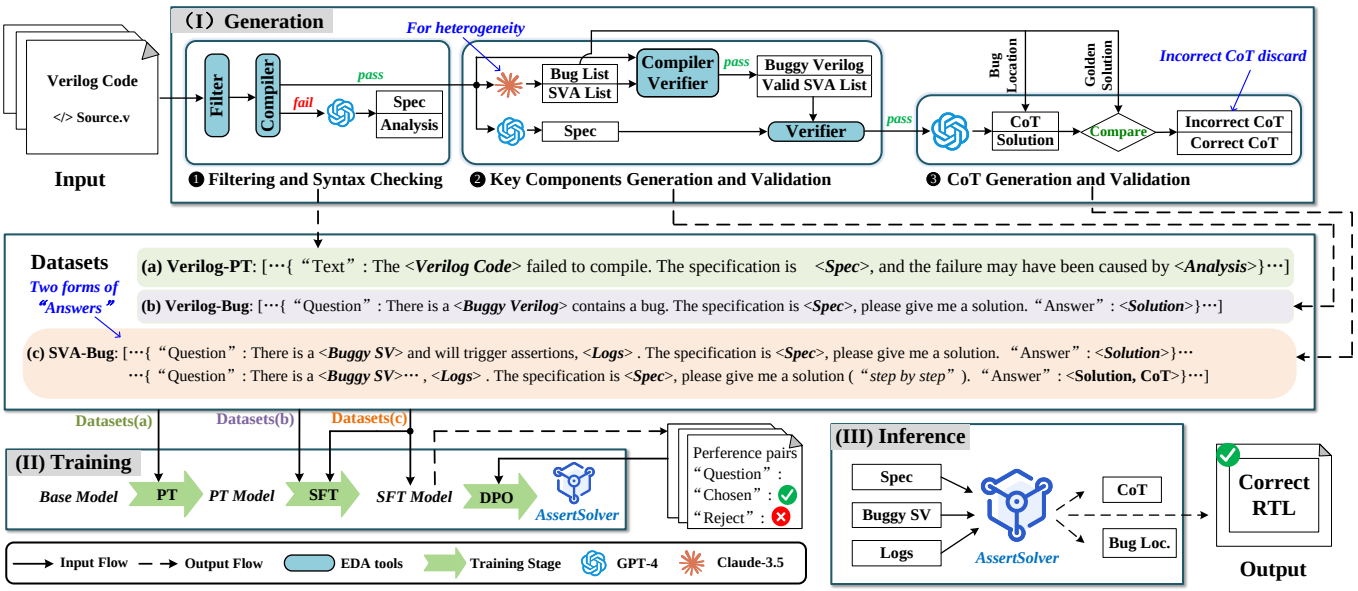


Fig. 2: AssertSolver overview: **(I)** shows the process for augmenting the training data. **(II)** describes the training strategy, consisting of pretraining (PT), Supervised Fine-Tuning (SFT), and Direct Preference Optimisation (DPO) which is used to guide learning from error responses to challenging cases. In the inference phase **(III)**, AssertSolver locates and fixes the bug based on the specification (Spec), buggy SystemVerilog (SV) code, and logs, while also providing bug location (Bug Loc.) and an explanation (CoT).

II. Data Augmentation

To effectively train an LLM for solving assertion failures, it is essential to have access to a comprehensive training dataset. While current datasets contain a significant number of Verilog code samples suitable for preliminary pretraining, they lack key components required for fine-tuning the model to this specific task. These components include design specifications, SVAs, details of bugs that could invalidate these assertions, and their corresponding verification outcomes. To address this gap, we propose a method that integrates LLMs and EDA tools to augment existing open-source Verilog code, thereby generating the necessary components for solving assertion failures. Additionally, to help users understand the model's problem-solving process, we have incorporated a Chain of Thought (CoT) [21] in the dataset. This inclusion is designed to enhance the transparency of the model's decision-making process. In this work, we augmented an open-source dataset [22] with 108,971 Verilog code samples from Hugging Face in the following 3 stages, as shown in Fig. 2-**(I)**.

Stage ❶: Filtering and Syntax Checking. Given the Verilog code samples from [22], we employed scripts to filter out code that exhibited certain undesirable characteristics. Specifically, we excluded code samples based on the following criteria:

1) Incomplete code that lacks either the '*module*' or '*endmodule*';
2) Code comprising only initialisation or assignment statements, with no functional logic;
3) Duplicated code segments.

Then, each remaining piece of code underwent a syntax check through the Icarus Verilog compiler [23]. Code with detectable syntax errors (e.g., those that failed the compilation), were excluded from the following process, preventing their advancement to the following stages where assertions could be triggered. However, these incorrect code samples were not discarded; instead, they are included in the **Verilog-PT** dataset as they offer valuable structural insights into Verilog code. After the syntax check, GPT-4 was employed to generate a design specification (Spec) for all code samples, but only those that failed compilation received further analysis. The analysis aims to provide an explanation for the causes of the syntax errors, contributing to the formation of the **Verilog-PT** dataset, as shown in Fig. 2 dataset (a). The **Verlog-PT** dataset, containing 22,646 entries, which is utilised in section III-A.

Stage ❷: Key Components Generation and Validation. For successfully compiled code, we employed Claude-3.5 to generate random bugs and SVAs. The use of Claude-3.5, as opposed to GPT-4, was intended to leverage the heterogeneous design of these models, thereby helping GPT-4 avoid falling into error traps during self-validation in **Stage ❸**.

To mitigate the impact caused by the hallucinations of LLMs [24]–[26], we utilised EDA tools at this stage to validate the correctness of the generated components. We employed a two-step verification process by integrating the compiler with the verifier, SymbiYosys [27]. Each generated SVA was inserted into the corresponding Verilog code and verified using SymbiYosys to ensure the SVAs' validity. Furthermore, we employed the compiler again to identify and eliminate syntax errors introduced during the random bug generation process. The remaining bugs were injected individually into the Verilog code, and each modified version was then verified with the corresponding SVAs using SymbiYosys to ensure that the bug-SVA pair caused assertion failures and to generate logs.

Approximately 90% of the bugs and SVA pairs associated with assertion failures were selected and processed in **Stage ❸**. The remaining 10% is reserved for testing, as detailed in Section IV-C. This selection process is structured to ensure that the training and test datasets are completely separate, involving the following steps:

1) Organise the buggy Verilog code into categories based on the length of the code, with bins defined as: $(0, 50]$, $(50, 100]$, $(100, 150]$, $(150, 200]$, and $(200, +\infty)$;
2) Enumerate the unique module names within each bin;
3) Uniformly select 90% of the module names and their corresponding buggy SystemVerilog (SV) code, Spec, and logs for inclusion in the training dataset.

Bugs that did not cause assertion failures, potentially due to insufficient SVA coverage, were retained as they represent functional issues in the original Verilog code. These bugs, together with the Spec, Verilog code, and solution, were organised into the **Verilog-Bug** dataset. This dataset, as shown in Fig. 2 dataset (b), contains 36,650 entries and is used in section III-B to enhance the model's understanding of the verification process.

Stage ❸: CoT Generation and Validation. To enhance the transparency of our model, we incorporated CoTs into the training data. This integration involves using GPT-4, where we provided Spec, buggy SV code, logs, and the bug location. The task for GPT-4 is to generate a CoT that articulates its reasoning behind identifying the erroneous code and suggesting correction.

Subsequently, we used a script to validate these CoTs by comparing GPT-4's output with the 'golden solution' obtained from Stage ❷. The 'golden solution' was derived from the initial buggy code and its correct counterpart. If GPT-4 identified an error and proposed correction align with it, we consider the CoT to be correct.

In total, approximately 74.55% of the generated CoTs were identified to be correct. Depending on the correctness of these CoTs, we organised two types of entries into the **SVA-Bug** dataset, which totals 7,842 entries as shown in Fig. 2 dataset (c). For entries where the CoT is correct, the 'Question' section includes the phrase '*step by step*', and the 'Answer' details both the buggy line and its corrected code, along with the CoT. Otherwise, the 'Answer' only includes the buggy line and the correct code. This dataset, designed specifically to equip the model with the ability to solve assertion failures and explain the reasoning process, is utilised in section III-B.

III. TRAINING STRATEGY

To maximise the potential of the dataset generated in Section II, we implemented a tailored training strategy for AssertSolver, as shown in Fig. 2-**(II)**. Our strategy started with a foundational **pretraining (PT)** phase using the base model, Deepseek-Coder-6.7b, on the **Verilog-PT** dataset. This step was to strengthen the model's understanding of Verilog code constructs and design specifications. Following this, we applied **Supervised Fine-Tuning (SFT)** to fine-tune the AssertSolver on the **SVA-Bug** and **Verilog-Bug** datasets. This step was designed to equip the model with the necessary skills for solving assertion failures, and to expand its capability for generalising across related Verilog debugging tasks. To further refine the model performance, we revisited unresolved assertion failure cases from the SFT phase, particularly those with incorrect responses, and employed **Direct Preference Optimisation (DPO)** to enable our model to learn from these error responses to challenging cases.

A. Pretraining

Pretraining is essential for LLMs, especially when preparing them to handle specialised languages such as Verilog and SystemVerilog. It is important to infuse domain-specific knowledge during this stage, which forms a robust base of understanding before any targeted fine-tuning starts. Recent research [14], [28] supports that continual pretraining on domain-specific dataset, including unlabeled Verilog code and syntactically similar language like C/C++ code, can substantially boosts the base model's understanding of hardware description languages (HDLs) and improve its performance in downstream tasks. In line with these insights, we implemented continual pretraining with the **Verilog-PT** dataset, comprising the Verilog code that failed in compilation along with its corresponding specifications and analyses of compilation failures. The base model, Deepseek-Coder-6.7b, has been preliminarily trained on a large programming corpus and is lightweight, making it ideal for this application. This focused pretraining strategy is essential as the timing and concurrency property in HDLs differ significantly from software programming paradigms.

During pretraining, each sample $x^{(i)}$ in the **Verilog-PT** dataset $D_{\text{PT}} = \{x^{(i)}\}_{i=1}^{N}$ is treated as a sequence of tokens. These tokens serve as the basic units in natural language processing tasks, allowing LLMs to leverage reasoning over them for next-token predication, which in turn facilitates text generation [29]–[32]. The sequence for each sample is expressed as $x^{(i)} = (w_1^{(i)}, w_2^{(i)}, \ldots, w_{T^{(i)}}^{(i)})$, where $w_t^{(i)}$ denotes the t-th token and $T^{(i)}$ the total number of tokens in the sequence. The training objective in this stage focuses on minimising the negative log-likelihood loss across the dataset:

$$L_{\text{PT}}(\theta) = -\sum_{i=1}^{N}\sum_{t=1}^{T^{(i)}} \log P(w_t^{(i)} \mid \text{context}_t^{(i)}; \theta)$$

Here $\text{context}_t^{(i)}$ refers to the series of preceding tokens which serve as the basis for predicting $w_t^{(i)}$, and $P(w_t^{(i)} \mid \text{context}_t^{(i)}; \theta)$ represents the probability of predicting t-th token based on this context, as determined by the model's parameter θ. This pretraining lays the groundwork for the subsequent fine-tuning process and equips the model capable of handling tasks within the hardware design domain.

B. Supervised Fine-Tuning (SFT)

Following the unsupervised pretraining phase, where the model primary learning was to predict the next token, Supervised Fine-Tuning (SFT) aims to shift the model's capabilities. This shift moves the focus from mere text continuation to solving specific question-answering challenges presented by assertion failures, which require precise and supervised responses.

To this end, we used the **SVA-Bug** dataset, which includes Spec, buggy SV code and logs. These elements are organised into the model's input x , as shown in the 'Question' section in Fig. 2 dataset (c). The model output the 'Answer' y must include, at a minimum, the bug line snippet and the corresponding correct code. Additionally, if the CoT is verified as correct in **Stage ❸** of section II, it is also included in y, enhancing the answer with detailed reasoning steps, marked by the keyword 'step by step' in x. Furthermore, we integrated the **Verilog-Bug** dataset as an auxiliary task to further enrich the training data with a broader spectrum of Verilog debugging scenarios. The data pairs $\langle x, y \rangle$ from this dataset are structured to provide the model with both the buggy Verilog code in the input 'Question' and the repair plan in the 'Answer', which lists the buggy line and alongside the corrected version, as shown in Fig. 2 dataset (b). This combination of datasets in the SFT process ensures a comprehensive learning experience for the model.

The objective of SFT is to refine the model's ability to predict the next token in $y^{(i)}$ based on the contextual interplay between the input $x^{(i)}$ and the sequence of previously generated tokens $y_{<t}^{(i)}$. This approach is designed to train the model in recognising and replicating the correct relationships between given 'Question' and 'Answer':

$$L_{\text{SFT}}(\theta) = -\sum_{i=1}^{N}\sum_{t=1}^{T_y^{(i)}} \log P(y_t^{(i)} \mid y_{<t}^{(i)}, x^{(i)}; \theta)$$

where $y_t^{(i)}$ denotes the t-th token in the output sequence $y^{(i)}$, and $y_{<t}^{(i)}$ represents the sequence of tokens preceding $y_t^{(i)}$. The likelihood $P(y_t^{(i)} \mid y_{<t}^{(i)}, x^{(i)}; \theta)$ indicates the probability of predicting the token $y_t^{(i)}$ given the $y_{<t}^{(i)}$ and the input context $x^{(i)}$, as governed by the model parameters θ. The SFT allows the model to produce the answer in the expected format and to develop a deeper understanding of the underlying hardware description language.

TABLE I: Bug types leading to assertion failures and examples

Type	Description	Expected Form	Unexpected Form	Assertion†
Direct	Bug signal appears directly in the assertion.	out <= **in;**	out <= **in + 1;**	assert(out == **in**)
Indirect	Bug signal does not appear directly in the assertion.	**temp <= in;** out <= temp;	**temp <= in + 1;** out <= temp;	assert(out == in)
Var	Incorrect variable name or type.	out = **in**;	out = **input**;	–
Value	Incorrect variable values, constants, or signal bit widths.	out = **4'b1010;**	out = **4'b1110;**	–
Op	Misuse of operators.	out = a **|** b;	out = a **&** b;	–
Cond	Bug in conditional statement (e.g., *if*, *always*).	**if (valid)** out <= in;	**if (!valid)** out <= in;	–
Non_cond	Bug unrelated to conditional statements.	if (valid) **out <= in;**	if (valid) **out <= input;**	–

† The distinction between *Direct* and *Indirect* type depends on whether the assertion failure is caused by the directly protected signal. Other types of bugs may cause assertion failures but are not necessarily directly reflected in the assertions.

TABLE II: Distribution of SVA-Bug and SVA-Eval across code length intervals and bug types (counts of instances)

Length Interval	(0, 50]	(50, 100]	(100, 150]	(150, 200]	(200, +∞)
SVA-Bug	3400	2444	921	431	646
SVA-Eval	431	260	102	58	64
Bug Type	Direct	Indirect	Var	Value	Op
SVA-Bug	5478	2364	546	5104	2254
SVA-Eval	615	300	47	601	274
Bug Type	Cond	Non_cond	–	–	–
SVA-Bug	1573	6269	–	–	–
SVA-Eval	204	711	–	–	–

C. Learning from Error Responses to Challenging Cases

At the SFT stage, AssertSolver is primarily exposed to correct responses, which limits its ability to process or learn from errors - a critical aspect of human learning. As noted in research by [33], [34], effective learning involves not only assimilating correct responses but also reflecting on and learning from mistakes to prevent future errors. Inspired by recent research [35]–[37], we propose to equip AssertSolver with the ability to learn from its errors.

To facilitate this, we evaluate the SFT model using all samples within the **SVA-Bug** dataset. Each sample, as illustrated in Fig. 2 dataset (c), includes a 'Question' section, which serves as the model's input. For each input, the model generates 20 responses. Correctness is then evaluated by comparing the buggy line suggested by the model with the correct 'Answer' in the dataset. Samples yielding at least one incorrect response among these 20 outputs are categorised as *challenging cases*, representing instances where the model struggles despite prior exposure to correct solutions. In each challenging case, the 'Question' is denoted as x and the correct 'Answer' as p. The incorrect responses to x are denoted as $n[k]$, where $k < 20$. We abstract them into triples: $D_{\text{DPO}} = \{(x^{(i)}, p^{(i)}, n[k]^{(i)})\}_{i=1}^{N}$, where N denotes the number of *challenging cases*. For this preference dataset D_{DPO}, the objective is to train the model to prioritise generating the correct response $p^{(i)}$ over the error responses $n[k]^{(i)}$ for each $x^{(i)}$. This goal is achieved through the DPO loss function:

$$L_{\text{DPO}} = -\mathbb{E}_D \left[\log \sigma \left(\beta \log \frac{\pi_\theta(p^{(i)}|x^{(i)})}{\pi_{\text{ref}}(p^{(i)}|x^{(i)})} - \beta \log \frac{\pi_\theta(n[k]^{(i)}|x^{(i)})}{\pi_{\text{ref}}(n[k]^{(i)}|x^{(i)})} \right) \right]$$

In this function, π_θ and π_{ref} represent the current model and the reference model (SFT model, in this context). The terms $\pi_\theta(p^{(i)}|x^{(i)})$ and $\pi_{\text{ref}}(p^{(i)}|x^{(i)})$ denote the likelihood of generating the correct response $p^{(i)}$ given input $x^{(i)}$ under the respective models. Similarly, $\pi_\theta(n[k]^{(i)}|x^{(i)})$ and $\pi_{\text{ref}}(n[k]^{(i)}|x^{(i)})$ correspond to the probabilities of generating the k-th error response $n[k]^{(i)}$ for the same input. The log-ratios of these probabilities quantify the divergence between the two models. The difference between the log-ratios captures the preference of π_θ for generating the correct response $p^{(i)}$ over the error response $n[k]^{(i)}$, guiding the model producing correct answers. The scaling factor β, set to 0.1, controls the weight of the log-ratio terms, and the sigmoid function σ maps the log-ratio values to a probability in the range $[0, 1]$, facilitating smooth learning while ensuring training stability. This approach allows AssertSolver not only to learn from errors but also to improve its response accuracy in challenging cases, leading to more robust decision-making capabilities.

IV. Evaluation

To evaluate the effectiveness of our trained model, we conduct extensive experiments designed to answer four key research questions. This section outlines the dataset property, benchmark and SOTA counterparts, evaluation metrics, and implementation details.

A. Research Questions

The evaluation is structured to investigate the following four research questions:

- **RQ1:** How does the incorporation of learning from error responses influence the performance of the model as measured by metrics of *pass@1* and *pass@5*?
- **RQ2:** How does the effectiveness of our model in solving assertion failures compare to that of other SOTA LLMs or models of similar complexity?
- **RQ3:** How does the model's performance vary when addressing randomly generated bugs versus human-crafted cases?
- **RQ4:** How is the model's performance impacted by design variability, specifically in relation to different bug types and variations in code length?

B. Dataset Property

The challenge of solving assertion failures varies considerably across different bug types and code lengths. Some categories are inherently more complex than others. For example, timing-related bugs that do not directly trigger assertion failures require deep reasoning and analysis, making them more complex. Similarly, longer code lengths may increase the complexity of debugging, as they often involve more intricate logic and a higher potential for subtle errors. Recognising this, we classified the types of bugs leading to assertion failures, as shown in Table I. We also analysed the code lengths and the number of instances falling into each identified bug category within our training and testing datasets, as illustrated in Table II.

TABLE III: Model performance as pass@k (grey shading: the best performance across models).

Metric	Base Model	SFT Model	AssertSolver
Pass@1	4.35%	84.66%	88.54%
Pass@5	15.62%	91.64%	90.00%

C. New Open-Source Benchmark and SOTA Counterparts

Given the lack of open-source benchmarks for evaluating tasks related to solving assertion failures, we have developed a benchmark, **SVA-Eval**, to address this gap. This benchmark consists of 877 samples generated by LLMs (**SVA-Eval-Machine**), as described in Section II, and 38 manually curated samples (**SVA-Eval-Human**) derived from the RTLLM dataset [38], ensuring both the scale of the dataset and the inclusion of real-world scenarios. Each sample in **SVA-Eval** includes the Spec, buggy SV code, logs, and correct solutions, providing a comprehensive resource for evaluation. The dataset is publicly available on **https://github.com/SEU-ACAL/reproduce-AssertSolver-DAC-25** to support further research.

For the comparison between AssertSolver and the current SOTA LLMs, we have chosen several commercially available closed-source models as benchmarks. These include Claude-3.5, GPT-4 [39], and OpenAI's latest o1-preview, all of which have demonstrated exceptional capabilities in RTL generation and debugging tasks. Furthermore, we have extended our comparative analysis to include open-source models such as CodeLlama-6.7b [40], Llama-3.1-8b, and our base model, Deepseek-Coder-6.7b [20]. This inclusive approach aims to provide a comprehensive overview of AssertSolver's performance across a spectrum of platforms and development environments.

D. Evaluation Metrics

To evaluate the performance of our LLM in solving assertion failures, we employ the ***pass@k*** metric, widely used in the evaluation performance of hardware designs [22], [41], [42]. This metric quantifies the effectiveness of LLMs by measuring their ability to generate correct solutions for each buggy SV code that causes assertion failures. For each instance, the model is provided with the buggy SV code alongside its corresponding specifications and logs, from which it generates n possible solutions. We then assess these solutions, deeming c of them effective if they successfully solve the assertion failure. This approach offers an unbiased estimate of the likelihood that at least one of the top k selections will address the problem, as shown by the following equation:

$$pass@k = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

In this study, we set $n = 20$ and $k = \{1, 5\}$. ***Pass@1*** assesses the model's accuracy and consistency by requiring the correct solution to be produced on the first attempt. An improvement in *pass@1* suggests that the model is becoming more adept in identifying and responding accurately to the bugs, thus likely improving its precision for such tasks. ***Pass@5*** evaluates whether model can provide at least one correct solution within five attempts. An increase in *pass@5* indicates that there is an enhancement in the model's ability to generate diverse solutions, reflecting its flexibility in problem-solving.

E. Implementation Details

Training. We fine-tuned the Deepseek-Coder-6.7b model using 8 A800-80G GPUs and accelerated the process with DeepSpeed ZeRO-3 [43]. We opted for full-parameter fine-tuning to achieve optimal performance and set an initial learning rate of 10^{-4} for pretraining and SFT, incorporating a warm-up phase during the first 10% of training steps. For DPO, a lower learning rate of 10^{-6} was used, as it focuses on learning the difference between correct and incorrect answers, rather than directly optimizing for explicit answers.

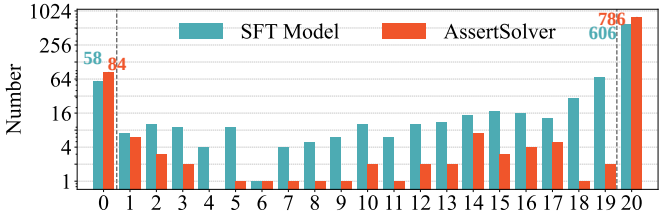


Fig. 3: Histogram of correct answers across 20 responses. (x-axis: c (number of correct solutions in 20 responses))

Inference. We combined the Spec, buggy SV code, and logs from the benchmark, requiring LLMs to return responses in a JSON format with a candidate buggy line, suggested fix, and CoT (Fig. 2-**(III)**). The temperature was 0.2 for consistent yet diverse outputs, except for o1-preview, which has a fixed temperature that cannot be adjusted through the application programming interface.

V. Results and Analysis

RQ1: We compared three models: the base model, the SFT model, and AssertSolver, using the **SVA-Eval** dataset. As shown in Table III, the base model showed the *pass@1* and *pass@5* rates below 5% and 16%, respectively. In contrast, both the SFT model and the AssertSolver, which underwent the process of pre-training and fine-tuning, consistently achieved more than 80% *pass@1* and *pass@5*. This presents a significant performance improvement, with over a 16-fold improvement in *pass@1* and a 5-fold increase in *pass@5*.

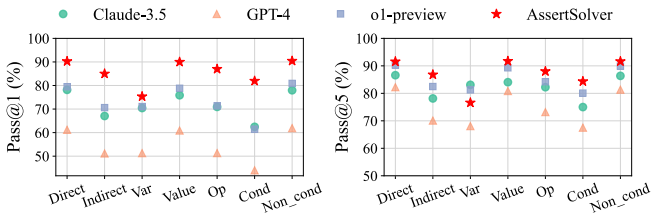
In evaluating the performance between the SFT model and the AssertSolver, a clear difference is observed. The AssertSolver demonstrated an improvement in *pass@1* performance, increasing from 84.66% to 88.54%. However, this was accompanied by a slight decline in *pass@5* performance, from 91.64% to 90%, compared to the SFT model. As outlined in Section IV evaluation metric, *pass@1* and *pass@5* reflect distinct characteristics of the underlying model. This observation suggests that although the model's precision improves with additional training on errors from challenging cases, its ability to generate a diverse range of solutions decreases.

Further analysis examined the performance of 915 test cases in the **SVA-Eval** dataset. In each case, it generated $n = 20$ possible solutions. We evaluated these solutions, deeming c of them effective. These test cases were then categorised into 21 distinct outcomes ranging from '$c = 0$' (indicating no effective solutions) to '$c = 20$' (where all solutions were effective). Intermediate values suggested varying levels of success and correlated with increased uncertainty as shown in Fig.3. In the figure, the AssertSolver generally outperforms the SFT model in deterministic scenarios (i.e., '$c = 0$' and '$c = 20$') but fell short in non-deterministic ranges. This finding, as supported by Table III, indicates that while adding challenging cases enhances the model's precision, it may limit the diversity of the solutions.

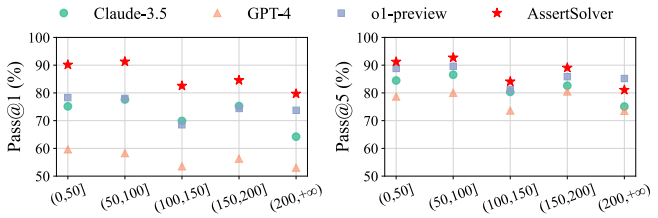
RQ2: For this research question, we compared the performance of AssertSolver with leading commercial and open-source LLMs, including the recently released o1-preview from OpenAI, focusing on *pass@1* and *pass@5* metrics within the **SVA-Eval** dataset. These comparisons are detailed in Table IV. AssertSolver outperforms all other models, achieving more than 80% in the *pass@1* metric (reaching 88.54%) and 90% in *pass@5*, while the second-best model,

TABLE IV: Performance comparison between AssertSolver and other LLMs (grey shading: the best performance across models).

Model	SVA-Eval-Machine		SVA-Eval-Human		SVA-Eval	
	pass@1 (%)	*pass@5 (%)*	*pass@1 (%)*	*pass@5 (%)*	*pass@1 (%)*	*pass@5 (%)*
Claude-3.5	74.86	84.10	66.58	77.48	74.52	83.83
GPT-4	58.04	78.45	54.74	74.01	57.90	78.27
o1-preview	76.96	87.73	67.50	87.94	76.57	87.74
Deepseek-coder-6.7b	4.41	15.85	2.89	10.27	4.35	15.62
CodeLlama-7b	5.95	17.06	4.47	12.85	5.89	16.89
Llama-3.1-8b	20.18	32.41	14.08	24.48	19.92	32.08
AssertSolver	89.04	90.38	76.97	81.35	88.54	90.00



(a) Performance comparison across different bug types



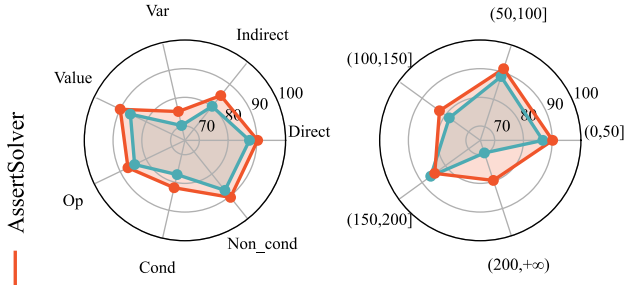
(b) Performance comparison across different code length intervals

Fig. 4: Comprehensive comparison with closed-source LLMs.

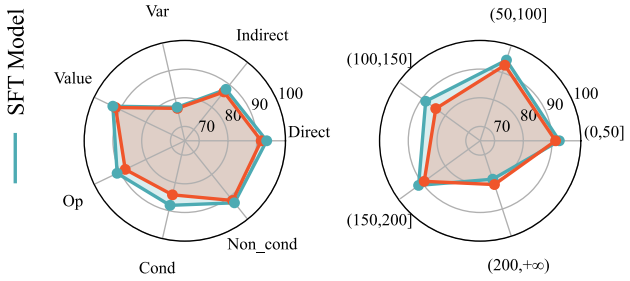
the o1-preview, scored 76.57% and 87.74% respectively. Further analysis, divided by the methods used to generate bugs, indicated that AssertSolver consistently performs best in all categories, except for *pass@5* in the **SVA-Eval-Human**. Considering the results from RQ1, where the AssertSolver demonstrated a preference for precision over diversity—thereby trading off performance in *pass@1* for *pass@5*—these outcomes align with expectations.

RQ3: As shown in Table IV, we observed that the performance for both *pass@1* and *pass@5* in **SVA-Eval-Human** consistently underperforms compared to **SVA-Eval-Machine**, with the exception of the *pass@5* metric on the o1-preview model. Across all tested models, there was an average relative decline of approximately 19% in *pass@1* and 15% in *pass@5*, which were calculated by averaging the ratio of the *pass@1* and *pass@5* rates between the **SVA-Eval-Machine** and **SVA-Eval-Human** datasets. This observation suggests there may be a systemic difference between machine and human-generated bugs, but requiring further investigation to confirm.

RQ4: To answer this research question, we evaluated AssertSolver's performance across different bug types and code lengths against closed-source LLMs, as shown in Fig. 4. AssertSolver consistently surpasses the performance of compared LLMs in *pass@1* across all tested scenarios and outperforms in *pass@5* in 10 out of 12 (83%) scenarios. Despite slight underperformance in the '*Var*' and in code length exceeding 200, AssertSolver demonstrates better results in all remaining cases. Particularly, for shorter code (under 100 lines) and the bugs classified as '*Direct*', '*Value*', and '*Non_cond*', both



(a) Pass@1 under different scenarios



(b) Pass@5 under different scenarios

Fig. 5: The performance of STF Model and AssertSolver in various bugs types and code length intervals.

pass@1 and *pass@5* reached over 90%, showcasing AssertSolver's effectiveness across various design scenarios.

Further analysis, as shown in Fig. 5, highlights how learning from error responses to challenging cases improves the *pass@1* across nearly all scenarios, particularly for code lengths exceeding 200 lines, with the exception of code between 150-200 lines. While there is a slight decrease in *pass@5* in these instances, this drop is not unexpected. AssertSolver prioritises precision over diversity, leading to performance that inherently favours *pass@1* results.

VI. Conclusion

In this work, we present AssertSolver, the first open-source LLM tailored to resolve assertion failures in RTL design. To address data underrepresentation, we propose an automatic data augmentation method that generates diverse failure scenarios, along with a novel training strategy that enables learning from both augmented data and challenging cases. Our experimental results show that AssertSolver achieves 88.54% pass@1 and 90.00% pass@5 on a 900+ instance test set, outperforming the o1-preview by 11.97% and 2.26%, respectively. By learning from challenging cases, AssertSolver further boosts its pass@1 performance, underscoring the model's suitability for applications that require high reliability and consistency. We have made AssertSolver publicly available for early adoption and released our evaluation benchmark. This work demonstrates that domain-specific fine-tuning of LLMs, combined with effective data augmentation and training strategies, can significantly advance the automation of solving assertion failures in hardware design.

VII. Acknowledgement

We appreciate the reviewers for their helpful feedback. This work is supported by the National Key Research and Development Program (Grant No. 2024YFB4405600), the National Natural Science Foundation of China (Grant No. 62472086), the Basic Research Program of Jiangsu (Grant No. BK20243042), the Science and Technology Major Special Program of Jiangsu (No. BG2024010), and the Start-up Research Fund of Southeast University (Grant No. RF1028624005).

REFERENCES

[1] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[2] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[3] H. Witharana, Y. Lyu, S. Charles, and P. Mishra, "A survey on assertion-based hardware verification," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–33, 2022.

[4] R. K. Ranjan, C. Coelho, and S. Skalberg, "Beyond verification: Leveraging formal for debugging," in *Proceedings of the 46th Annual Design Automation Conference*, 2009, pp. 648–651.

[5] E. Seligman, T. Schubert, and M. A. K. Kumar, *Formal verification: an essential toolkit for modern VLSI design*. Elsevier, 2023.

[6] K. Maddala, B. Mali, and C. Karfa, "Laag-rv: Llm assisted assertion generation for rtl design verification," in *2024 IEEE 8th International Test Conference India (ITC India)*. IEEE, 2024, pp. 1–6.

[7] S. S. Miftah, A. Srivastava, H. Kim, and K. Basu, "Assert-o: Context-based assertion optimization using llms," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 233–239.

[8] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, H. Zhang, and Z. Xie, "Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms," *arXiv preprint arXiv:2402.00386*, 2024.

[9] V. Pulavarthi, D. Nandal, S. Dan, and D. Pal, "Assertionbench: A benchmark to evaluate large-language models for assertion generation," *arXiv preprint arXiv:2406.18627*, 2024.

[10] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, "Using llms to facilitate formal verification of rtl," *arXiv e-prints*, pp. arXiv–2309, 2023.

[11] C. Sun, C. Hahn, and C. Trippel, "Towards improving verification productivity with circuit-aware translation of natural language to systemverilog assertions," in *First International Workshop on Deep Learning-aided Verification*, 2023.

[12] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–8.

[13] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[14] N. Wang, B. Yao, J. Zhou, X. Wang, Z. Jiang, and N. Guan, "Large language model for verilog generation with golden code feedback," *arXiv preprint arXiv:2407.18271*, 2024.

[15] C.-T. Ho, H. Ren, and B. Khailany, "Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool," *arXiv preprint arXiv:2408.08927*, 2024.

[16] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[17] K. Xu, J. Sun, Y. Hu, X. Fang, W. Shan, X. Wang, and Z. Jiang, "Meic: Re-thinking rtl debug automation using llms," *arXiv preprint arXiv:2405.06840*, 2024.

[18] W. Fu, K. Yang, R. G. Dutta, X. Guo, and G. Qu, "Llm4sechw: Leveraging domain-specific large language model for hardware debugging," in *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2023, pp. 1–6.

[19] X. Yao, H. Li, T. H. Chan, W. Xiao, M. Yuan, Y. Huang, L. Chen, and B. Yu, "Hdldebugger: Streamlining hdl debugging with large language models," *arXiv preprint arXiv:2403.11671*, 2024.

[20] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[21] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[22] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[23] S. Williams and M. Baxter, "Icarus verilog: open-source verilog more than a year later," *Linux Journal*, vol. 2002, no. 99, p. 3, 2002.

[24] G. Perković, A. Drobnjak, and I. Botički, "Hallucinations in llms: Understanding and addressing challenges," in *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*. IEEE, 2024, pp. 2084–2088.

[25] G. P. Reddy, Y. P. Kumar, and K. P. Prakash, "Hallucinations in large language models (llms)," in *2024 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*. IEEE, 2024, pp. 1–6.

[26] S. Tonmoy, S. Zaman, V. Jain, A. Rani, V. Rawte, A. Chadha, and A. Das, "A comprehensive survey of hallucination mitigation techniques in large language models," *arXiv preprint arXiv:2401.01313*, 2024.

[27] C. Wolf *et al.*, "Symbiyosys," *URL: https://symbiyosys. readthedocs. io/.[Cited on page 6.]*, 2022.

[28] B. Yao, N. Wang, J. Zhou, X. Wang, H. Gao, Z. Jiang, and N. Guan, "Location is key: Leveraging large language model for functional bug localization in verilog," *arXiv preprint arXiv:2409.15186*, 2024.

[29] S. J. Mielke, Z. Alyafeai, E. Salesky, C. Raffel, M. Dey, M. Gallé, A. Raja, C. Si, W. Y. Lee, B. Sagot *et al.*, "Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp," *arXiv preprint arXiv:2112.10508*, 2021.

[30] M. Qi, Y. Huang, Y. Yao, M. Wang, B. Gu, and N. Sundaresan, "Is next token prediction sufficient for gpt? exploration on code logic comprehension," *arXiv preprint arXiv:2404.08885*, 2024.

[31] R. Sennrich, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[32] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa, "Byte pair encoding: A text compression scheme that accelerates pattern matching," 1999.

[33] N. Mercer, "Talk and the development of reasoning and understanding," *Human development*, vol. 51, no. 1, pp. 90–100, 2008.

[34] T. Reich, A. Kaju, and S. J. Maglio, "How to overcome algorithm aversion: Learning from mistakes," *Journal of Consumer Psychology*, vol. 33, no. 2, pp. 285–302, 2023.

[35] Y. Tong, D. Li, S. Wang, Y. Wang, F. Teng, and J. Shang, "Can llms learn from previous mistakes? investigating llms' errors to boost for reasoning," *arXiv preprint arXiv:2403.20046*, 2024.

[36] K. Chen, C. Wang, K. Yang, J. Han, L. Hong, F. Mi, H. Xu, Z. Liu, W. Huang, Z. Li *et al.*, "Gaining wisdom from setbacks: Aligning large language models via mistake analysis," *arXiv preprint arXiv:2310.10477*, 2023.

[37] S. An, Z. Ma, Z. Lin, N. Zheng, J.-G. Lou, and W. Chen, "Learning from mistakes makes llm better reasoner," *arXiv preprint arXiv:2310.20689*, 2023.

[38] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 722–727.

[39] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[40] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[41] H. Huang, Z. Lin, Z. Wang, X. Chen, K. Ding, and J. Zhao, "Towards llm-powered verilog rtl assistant: Self-verification and self-correction," *arXiv preprint arXiv:2406.00115*, 2024.

[42] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[43] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

A Mixed-Signal Photonic SRAM-based High-Speed Energy-Efficient Photonic Tensor Core with Novel Electro-Optic ADC

Md Abdullah-Al Kaiser*, Sugeet Sunder†, Ajey P. Jacob†, and Akhilesh R. Jaiswal*
*University of Wisconsin–Madison, WI, USA, †USC Information Sciences Institute, CA, USA
mkaiser8@wisc.edu, sunder@isi.edu, ajey@isi.edu, akhilesh.jaiswal@wisc.edu

***Abstract*—The rapid surge in data generated by Internet of Things (IoT), artificial intelligence (AI), and machine learning (ML) applications demands ultra-fast, scalable, and energy-efficient hardware, as traditional von Neumann architectures face significant latency and power challenges due to data transfer bottlenecks between memory and processing units. Furthermore, conventional electrical memory technologies are increasingly constrained by rising bitline and wordline capacitance, as well as the resistance of compact and long interconnects, as technology scales. In contrast, photonics-based in-memory computing systems offer substantial speed and energy improvements over traditional transistor-based systems, owing to their ultra-fast operating frequencies, low crosstalk, and high data bandwidth. Hence, we present a novel differential photonic SRAM (pSRAM) bitcell-augmented scalable mixed-signal multi-bit photonic tensor core, enabling high-speed, energy-efficient matrix multiplication operations using fabrication-friendly integrated photonic components. Additionally, we propose a novel 1-hot encoding electro-optic analog-to-digital converter (eoADC) architecture to convert the multiplication outputs into digital bitstreams, supporting processing in the electrical domain. Our designed photonic tensor core, utilizing GlobalFoundries' monolithic 45SPCLO technology node, achieves computation speeds of 4.10 tera-operations per second (TOPS) and a power efficiency of 3.02 TOPS/W.**

***Index Terms*—photonic memory, in-memory compute, photonic ADC, microring resonator, tensor core.**

I. Introduction

The rapid growth of data-intensive applications, such as artificial intelligence (AI), machine learning (ML), and big data analytics, highlights the memory-wall bottleneck in von Neumann systems, where frequent data transfers between the processor and memory reduce speed, bandwidth, and power efficiency [1]. Addressing this requires a shift in hardware design, exploring new computing paradigms that enhance computational throughput while tackling energy and latency concerns. In-memory computing (IMC), where computation occurs within memory to minimize data transfer delays, is one such approach [2], [3]. Other emerging paradigms, like non-volatile memory (NVM) computing [4], optical computing [5], and quantum computing [6], can also improve efficiency by bypassing traditional system limitations.

SRAM-based in-memory computing (IMC) macros offer advantages like unlimited endurance and compatibility with existing silicon foundries, but face challenges such as crosstalk between adjacent bitcells during simultaneous row activation, which can lead to read-disturb errors and high power consumption [7]–[9]. Non-volatile memory (NVM) devices like resistive RAM (RRAM) [10], [11], phase-change memory (PCM) [12], [13], and magneto-resistive RAM (MRAM) [14], [15] offer benefits such as non-volatility and lower power use, but have limitations. PCM and RRAM suffer from slow read/write speeds (nanoseconds to microseconds), high write energy, and limited endurance [11], [16]–[18], while MRAM, though faster, has a low on-off resistance ratio, making it error-prone and thermally unstable [16], [19]. Furthermore, both SRAM and NVM approaches depend on electrical interconnects, but as technology scales, these interconnects face significant challenges, including higher coupling capacitance and increased metal wire resistance that exacerbates signal delay and noise issues [20]–[23]. These factors collectively limit data throughput and increase energy consumptions for high-speed and energy-efficient computing systems.

In contrast, photonics systems leverage waveguides to confine and guide light, effectively addressing the scaling challenges faced by electrical interconnects. These systems enable low-loss optical data transfer over significant distances by utilizing carefully designed structures that exploit principles such as total internal reflection, index guidance, or photonic bandgap [24], [25]. Moreover, wavelength-division multiplexing (WDM) significantly enhances computational efficiency, bandwidth, and throughput by enabling simultaneous data transmission on multiple wavelengths, each carrying distinct information [26]. Various photonic IMC macros demonstrate ultra-high-speed, energy-efficient operations by encoding inputs as optical pulse intensity and controlling weights via the transmittance or phase of light [27]–[35]. While Mach-Zehnder interferometer (MZI)-based photonic compute cores allow rapid weight updates, their large device area limits scalability for matrix computations [32]–[34]. In contrast, phase-change material (PCM)-based systems offer scalability by controlling transmittance as a weight; however, they demand high write latency and energy, reducing efficiency in large-scale applications requiring frequent updates [28], [30], [31], [36]. Microring resonator (MRR)-based photonic tensor cores provide a compact footprint, enabling higher integration density and scalability critical for large-scale matrix compute cores [29], [35]. However, MRRs are susceptible to thermal and environmental fluctuations, which can be effectively mitigated

through thermal tuning using integrated heaters to stabilize operating conditions [37], [38]. Hence, MRR-based photonic IMC systems can deliver high-speed, energy-efficient, scalable, and compact solutions, advancing photonic computing for large-scale applications requiring frequent weight updates.

This work presents a mixed-signal, multi-bit, scalable differential photonic SRAM-embedded tensor core enabling high-speed, energy-efficient matrix multiplication and high-speed memory updates using fabrication-friendly silicon photonics components. A critical challenge in photonic IMC systems—efficient post-processing of computed analog results—is also addressed. Many photonic IMC macros often depend on off-chip processing, such as optical power measurements [28], [29], [34], electrical ADCs [33], or digital signal processors [30], which create performance bottlenecks that reduce speed and energy efficiency. To overcome these limitations, we propose a novel monolithic electro-optic ADC (eoADC) architecture. Unlike traditional high-speed flash ADCs [39], [40], which are power-intensive due to their thermometer-coded design requiring numerous comparator activations, or time-interleaved ADCs, which face synchronization issues and high power consumption [41]–[43], our eoADC employs a one-hot encoding method. This approach activates only a single thresholding block per conversion, minimizing energy consumption while maintaining flash ADC-level speeds. Additionally, this single-slice design can be extended using a time-interleaved configuration to further enhance speed. By integrating this scalable, energy-efficient eoADC into the photonic tensor core, we provide a seamless, end-to-end architecture compatible with electrical subsystems, advancing large-scale, high-performance computing for big data applications.

The main contributions of this paper are as follows:

1) We present a novel mixed-signal, multi-bit, and scalable photonic SRAM-augmented photonic tensor core, enabling high-speed, energy-efficient matrix multiplication for complex computational tasks.
2) The architecture supports memory updates at 20 GHz rate, ensuring high-speed operations for big data applications where datasets exceed memory array capacity and require frequent, rapid updates.
3) Additionally, we propose a novel 1-hot encoding electro-optic analog-to-digital converter (eoADC) architecture that utilizes fabrication-friendly silicon photonics components, achieving a speed of 8 GS/s with an energy consumption of 2.32 pJ per conversion.
4) Finally, our photonic SRAM-embedded, multi-bit, mixed-signal photonic tensor core with eoADC demonstrates a computation speed of 4.10 tera-operations per second (TOPS) and a power efficiency of 3.02 TOPS/W.

II. Photonic Compute Preliminaries

This section explores the core building blocks of the photonic tensor core, leveraging fabrication-friendly silicon photonics components such as waveguides, microring resonators (MRRs), photodiodes (PDs), optical power splitters (PS), and passive absorbers (A). Waveguides confine and guide light through high-refractive-index materials like silicon, with low-refractive-index cladding like silicon dioxide ensuring internal reflection. MRRs consist of a circular waveguide (ring) coupled to one or more straight bus waveguides [37]. When the resonance condition is met—i.e., when the optical path length of the ring is an integer multiple of the input optical wavelength—light couples into the ring; otherwise, it continues through the bus [38]. This resonance depends on factors such as the wavelength of the input light, the effective refractive index of the waveguide mode, and the ring length. Precise tuning of the resonance wavelength can be achieved by modulating the refractive index via an electrical bias across an integrated pn junction (utilizing the plasma dispersion effect). Photodiodes convert optical signals into electrical currents, power splitters divide light among multiple waveguides, and optical absorbers capture stray light to prevent reflections or crosstalk. These mature silicon photonics components utilized to design the photonic SRAM-augmented tensor core and electro-optic ADC, ensuring compatibility with existing silicon processes and integration into electronic platforms. Subsequent sections provide further detail on these photonic blocks.

A. Cross-coupled Differential Photonic SRAM Bitcell:

Fig. 1 presents the schematic of the photonic SRAM (pSRAM) bitcell. In this configuration, M1-M2 are microring resonators (MRRs), P1-P4 are photodiodes (PDs), PS1-PS3 are optical power splitters, and A1-A2 are passive optical absorbers to minimize unwanted reflections. An optical laser (λ_{IN}) is connected to the input power splitter (PS1), which directs the power to the input bus waveguides of two identical MRRs (M1 and M2). The wavelength λ_{IN} is selected to resonate with the MRRs when a voltage VDD is applied to them. The thru and drop bus waveguides of M1 (M2) are connected to the waveguides of photodiodes P1 (P3) and P2 (P4), respectively. The midpoints between photodiodes P1 and P2 (P3 and P4) are labeled as QB (Q), serving as the electrical storage nodes of the pSRAM. Node QB (Q) drives M2 (M1) through an electrical driver D1 (D2), creating a cross-coupled structure to hold the stored data. This arrangement forms a pSRAM latch capable of storing binary data at the storage nodes Q (data) and QB (complementary data) as long as

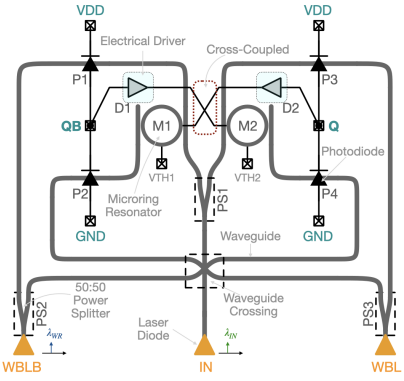


Fig. 1. Differential cross-coupled photonic SRAM bitcell.

both the optical bias (optical bias via IN) and electrical bias (VDD) are maintained. To retain data, the cross-coupled electro-optic structure must maintain stability. For instance, when Q = 1 (VDD) and QB = 0 (GND) are stored, these values must be preserved as long as the optical and electrical biases are applied. With Q = 1, M1 is tuned to resonance with λ_{IN}, coupling most of the light to P2 via its drop port. This generates a higher photocurrent in P2, creating a low-resistance path to GND and keeping QB at 0. Conversely, QB = 0 shifts the resonance wavelength of M2 away from λ_{IN}. This allows light to pass through M2's thru port to P3, maintaining a high photocurrent and keeping Q at VDD. The complementary states of M1 and M2, coupled with their respective photodiodes, create a positive feedback loop that maintains the stored data. The same mechanism applies in reverse to maintain Q = 0 and QB = 1, where M2 would be on-resonance and M1 off-resonance.

Data can be written into the pSRAM cell by applying differential optical power through the write bitline waveguides (WBL and WBLB). Starting with Q = 0 (GND) and QB = 1 (VDD), to switch the data to Q = 1 and QB = 0, higher optical power is supplied to the WBL waveguide while no power is given to the WBLB. The write optical power must exceed the input bias laser power for successful data flipping and can operate at a different/same wavelength, as photodiodes generally have a broadband response. This causes P3 (P2) to generate more current than P4 (P1), creating a low-resistance path to VDD (GND) for Q (QB), making Q rise to VDD and bringing M1 into resonance, and, QB drops to GND, stabilizing the state. The opposite state (Q = 0, QB = 1) can be written by reversing the optical power between the WBL and WBLB waveguides. More details about the structure and operation of the pSRAM bitcell can be found in [44].

B. Mixed-signal Multi-bit Photonic Vector Multiplication Compute Core:

Fig. 2 depicts the mixed-signal, multi-bit (n-bit) vector multiplication compute core, designed for vector-vector multiplication ($\mathrm{IN} = [\mathrm{IN}_1, \mathrm{IN}_2, \ldots, \mathrm{IN}_\mathrm{m}] \times \mathrm{W} = [\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_\mathrm{m}]$), where IN, w, and m represent the input vector, weight vector, and vector dimension, respectively. The core uses intensity-encoded optical pulses as analog inputs and multi-bit (n-bit) pSRAM (detailed in Section II-A) for weight storage. In the 1-bit mixed-signal multiplication example in Fig. 2, each unit includes a MRR controlled by a pSRAM bitcell storage node representing a 1-bit weight (w). The MRR's wavelength is tuned such that, when $\mathrm{w} = 0$, the incoming light is coupled inside the ring, resulting in no output at the thru port (output = 0). Conversely, for $\mathrm{w} = 1$, the MRR is off-resonance, allowing the light to pass through the thru port. This mechanism enables the multiplication of the analog input (IN) by the binary weight (w), producing an output of 0 or IN based on the weight value.

The analog intensity-encoded vector can be generated using an optical frequency comb, which produces multiple precisely spaced wavelengths [30], enabling parallel data transmission through Wavelength Division Multiplexing (WDM). The input

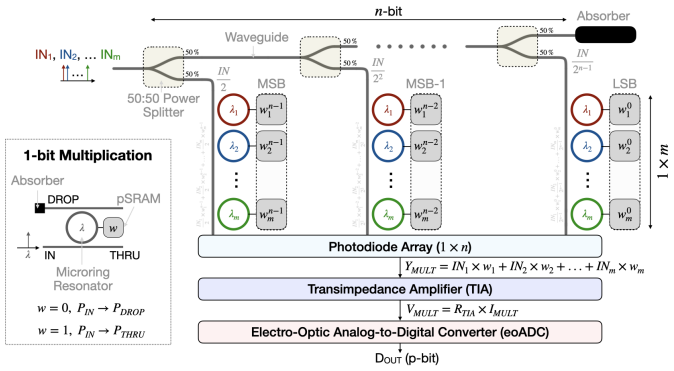


Fig. 2. Mixed-Signal multi-bit photonic vector multiplication compute core.

vector ($\mathrm{IN} = \mathrm{IN}_1, \mathrm{IN}_2, \ldots, \mathrm{IN}_\mathrm{m}$) of size m is transmitted through a single bus waveguide via WDM, with each input intensity encoded at a different wavelength (λ_1, λ_2, ..., λ_m). For n-bit MAC operations, n pSRAMs are assigned to each weight, with weights organized by bit significance ($\mathrm{w}_1 = \mathrm{w}_1^{n-1}\mathrm{w}_1^{n-2}\ldots\mathrm{w}_1^0$), where w_1^{n-1} is the most significant bit (MSB) and w_1^0 is the least significant bit (LSB). The analog inputs are distributed through n cascaded power splitters, generating binary-scaled input values ($\frac{\mathrm{IN}}{2}, \frac{\mathrm{IN}}{2^2}, \ldots, \frac{\mathrm{IN}}{2^{\mathrm{n}-1}}$) [45].

The binary-scaled analog signals are multiplied using a 1-bit multiplication structure, which consists of a MRR driven by a 1-bit pSRAM. Each ring is tuned to a specific input wavelength by modulating its length, so each performs multiplication on a corresponding input. For instance, the red, blue, and green rings in Fig. 2 perform 1-bit mixed-signal multiplication on IN_1, IN_2, and IN_m, respectively. WDM enables computing across multiple wavelengths within a single bus waveguide without crosstalk. The mixed-signal, wavelength-multiplexed multiplication results are combined within the waveguide and directed to a photodiode. The photodiode array, with n differential waveguides for n-bit multiplication, aggregates these signals to produce an output equal to the vector-vector multiplication of the analog inputs and n-bit weights. The results are then converted into digital bitstreams (p-bit) using a high-bandwidth transimpedance amplifier (TIA) and ADC.

C. 1-hot Encoding Electro-Optic ADC:

Fig. 3(a) shows the transmission spectra of a two-port MRR, where the resonance state determines whether the thru port receives light. The transmission characteristics are modulated by applying voltage across the pn junction, which controls the refractive index. Three transmission spectra are shown, color-coded as red, black, and blue, corresponding to different applied voltages ($\mathrm{V_{REF1}} > \mathrm{V_{REF2}} > \mathrm{V_{REF3}}$) at the p-terminal. In this simulation, $\mathrm{V_{IN}}$ is set to $\mathrm{V_{REF2}}$, with the resonance wavelength (λ_{IN}) selected when $\mathrm{V_{pn}} = 0$. At this condition, the MRR's thru port exhibits the lowest power at λ_{IN}, shown by the black curve. When the applied voltage is $\mathrm{V_{REF1}}$ or $\mathrm{V_{REF3}}$, the MRR remains off-resonance due to $\mathrm{V_{pn}} > 0$ or $\mathrm{V_{pn}} < 0$, resulting in higher output power at λ_{IN} ($> \mathrm{P_{REF}}$). As $\mathrm{V_{IN}}$ increases beyond $\mathrm{V_{REF2}}$, the spectra shift to longer wavelengths (red-shift), and when $\mathrm{V_{IN}}$ reaches $\mathrm{V_{REF1}}$, the blue

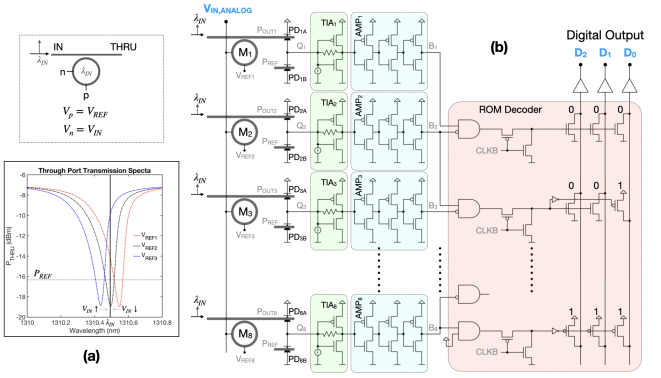


Fig. 3. (a) MRR transmission spectra as a function of the pn junction voltage, (b) 1-hot encoding electro-optic ADC architecture.

curve aligns with the black curve, showing lower output power at λ_{IN}. Similarly, decreasing V_{IN} shifts the spectra to shorter wavelengths (blue-shift), with the blue curve aligning with the black curve, exhibiting minimum power. This demonstrates that by applying specific reference voltages at the p-terminal and connecting the input to the n-terminal of the pn junction, the MRR can selectively resonate at the input wavelength when V_{IN} is close to the reference voltage, allowing for ADC quantization at a particular code value.

Leveraging the voltage-dependent notch-like response of the MRR thru port, we present a novel 1-hot encoding electro-optic ADC (eoADC) architecture, shown in Fig. 3(b). While the figure illustrates a 3-bit eoADC, the design is scalable to p-bit ADCs. This ADC converts an analog input voltage into 3-bit digital electrical bitstreams, using light as the state variable for high-speed thresholding. For a 3-bit ADC, 8 (for p-bit, 2^p) MRRs are used, with the n-terminals of the pn junctions connected to the input voltage and the p-terminals connected to different reference voltages. This ensures each MRR resonates within a specific input voltage range. For example, MRR M_1 exhibits minimal power at the thru port when the input voltage $V_{IN,ANALOG}$ is within 0 to $\frac{V_{FS}}{8}$, where V_{FS} is the full-scale ADC input range. Other reference voltages are similarly set to align each MRR to resonate at a distinct input voltage range. By exploiting the transmission spectra and resonance wavelength modulation—governed by the applied voltage across the pn junction—the eoADC achieves 1-hot encoding behavior, distinguishing it from the thermometer-coded approach typically seen in power-hungry flash ADCs.

A balanced photodiode structure is used as the opto-electric thresholding block, where the lower photodiode connects to a reference optical power (P_{REF}) and the upper photodiodes are linked to the thru ports of the MRRs, each calibrated to resonate at specific input voltage ranges. When an MRR is on-resonance due to the input voltage, the corresponding upper photodiode receives less optical power than the lower photodiode, causing the output node (Q_p) to discharge toward ground. The output voltage at Q_p is amplified through an inverter-based high-speed TIA and a cascaded voltage amplifier, converting the voltage change into a rail-to-rail swing [46]. This amplified signal (B_p) is sent to a ROM-based decoder circuit, which implements a ceiling function between adjacent channels for fast digital bitstream conversion. The ceiling function ensures robustness by resolving cases where the input voltage is at the midpoint of two voltage ranges, preventing simultaneous activation of two digital codes. This avoids static current flow through the decoder, enhancing reliability. The current demonstration utilizes a 3-bit ADC, leveraging the MRR in the GF45SPCLO node; however, higher precision can be achieved by optimizing devices, such as using high-Q MRRs, or by cascading multiple lower-bit ADCs with shift-and-add operations. Additionally, this ADC architecture can also be integrated utilizing time-interleaved structures to improve the operating speed.

III. Mixed-Signal Multi-bit Scalable Photonic Tensor Core

Fig. 4 illustrates a scalable 2D mixed-signal, multi-bit photonic tensor core architecture designed for matrix multiplication, which is achieved by tiling the vector multiplication compute core discussed in Section II-B. This core employs WDM for mixed-signal multiplication, requiring precise wavelength selection and careful tuning of the MRRs. A key design factor is the number of usable wavelength channels within the MRR's free spectral range (FSR). For instance, with a 9 nm FSR and 2 nm channel spacing, up to four wavelength channels can be effectively used without causing side-channel interference. Channel spacing can further be lowered to support more wavelength channels depending on the MRR transmission characteristics. In this work, four wavelengths (λ_1, λ_2, λ_3, λ_4) are assigned per vector compute macro, allowing for the multiplication of 1×4 input and weight vectors. Specifically, the input elements IN_1, IN_2, IN_3, and IN_4 are multiplied by MRRs tuned to the wavelengths λ_1, λ_2, λ_3, and λ_4, respectively, with control provided by the pSRAM bitcell arrays storing the corresponding weights w_1, w_2, w_3, and w_4. Although Fig. 4 shows a 3-bit weight precision, the precision can be enhanced by adding more MRRs and pSRAM bitcells. While the number of wavelengths per compute core can limit the vector size, the architecture can be scaled by replicating the vector compute macro to handle larger vectors ($1 \times m$). For

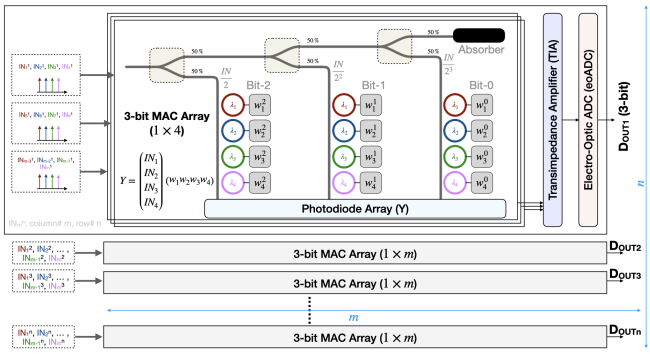


Fig. 4. Mixed-signal multi-bit scalable 2D photonic tensor core enabling matrix multiplication.

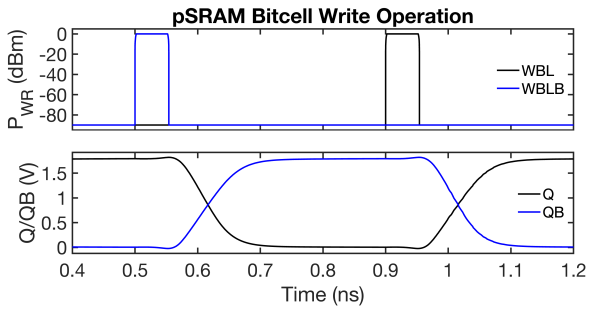


Fig. 5. Verification of weight configuration in pSRAM bitcell.

example, to perform 1×16 vector multiplications, four 1×4 vector compute macros can be used, with results obtained through current summation in the photodiodes. These results are then digitized using the eoADC, as explained in Section II-C. Replicating the $1 \times \mathrm{m}$ compute core n times allows multiplication for an $\mathrm{m} \times \mathrm{n}$ array.

IV. Simulation Results and Performance Analysis

This section presents the verification results and performance metrics of the building blocks in our proposed photonic tensor core, simulated using the monolithic GF45SPCLO technology node.

A. Weight Configuration in pSRAM Bitcell

To configure weight values, data 0 or 1 is written into the pSRAM bitcell by applying a differential optical pulse via the write bitlines. In Fig. 5, the top subplot shows the optical write laser input to the WBL and WBLB waveguides, using a 50 ps wide write pulse at 0 dBm power. The bottom subplot illustrates how storage nodes Q and QB respond to optical inputs at WBL and WBLB, respectively; a write pulse on WBL (or WBLB) sets Q (or QB) to 1. The stabilized (hold mode) Q and QB states after data flip are also shown. With a -20 dBm optical bias and a wall-plug efficiency of 0.23 [47] for the input and write laser source, the pSRAM consumes 0.5 pJ of energy per switching event (weight update) at a speed of 20 GHz.

B. Vector Multiplication Compute Core

Achieving WDM-based multiplication within a single waveguide (as discussed in Section II-B) requires precise selection of input and resonance wavelengths. The MRR, with a 7.5 μm ring radius and a 200 nm gap at the thru-port, achieves four distinct resonance wavelengths (λ_1, λ_2, λ_3, and λ_4) by adjusting the ring length by 0 nm, 68 nm, 136 nm, and 204 nm, respectively. With an FSR of 9.36 nm and a wavelength separation of 2.33 nm, minimal crosstalk is ensured. While more wavelengths could fit within the FSR by reducing the separation, this work focuses on using four wavelengths for multiplication on four inputs within the same waveguide.

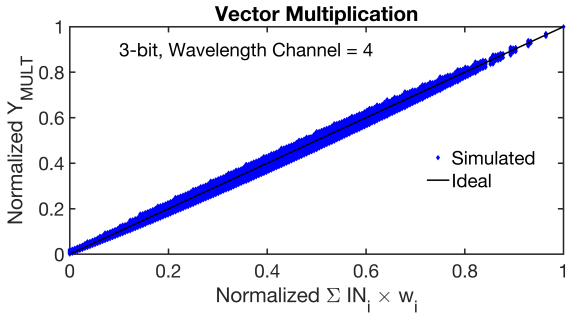


Fig. 7. Simulation results of multiplying two 1×4 vectors using 3-bit weight precision and four wavelength channels.

Fig. 7 illustrates the simulation results for multiplying two 1×4 vectors, where the input values are represented by analog light intensities, and the weights are encoded using 3-bit pSRAM bitcells. The simulation leverages four wavelength channels, though the GF45SPCLO node supports simulation for only one wavelength at a time. To address this, each wavelength channel is simulated separately, with all the MRRs included in the testbench to incorporate the inter-channel crosstalk, and the results are combined linearly through photodiode current summation to generate the final output. Ideally, the normalized photodiode current output should align linearly with the vector multiplication results, and the simulated outputs follow this trend, as shown in the figure.

C. Electro-optic ADC Verification

Fig. 8 shows the MRR thru-port transmission spectra ($\mathrm{P_{OUT}}$, as in Fig. 3) of the electro-optic ADC. Each MRR ($\mathrm{M_1}$ to $\mathrm{M_8}$) corresponds to a distinct reference voltage, creating dips in the transmission spectra within specific input voltage ranges. As the input voltage approaches a reference voltage,

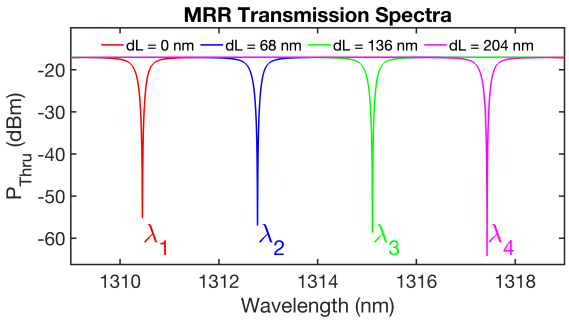


Fig. 6. Transmission spectra of the MRR as a function the ring adjustment length. Here, dL denotes the adjusted length from the base ring structure.

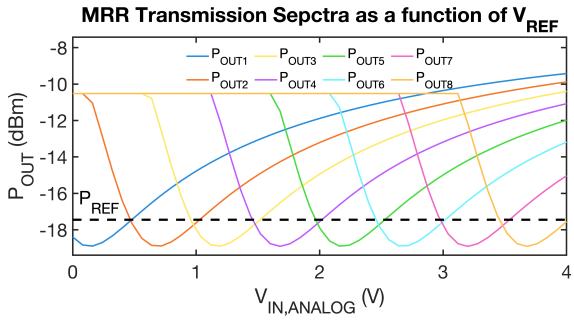


Fig. 8. MRR transmission spectra versus input analog voltage as a function of different reference voltages ($\mathrm{V_{REF}}$) exhibiting 1-hot encoding characteristics.

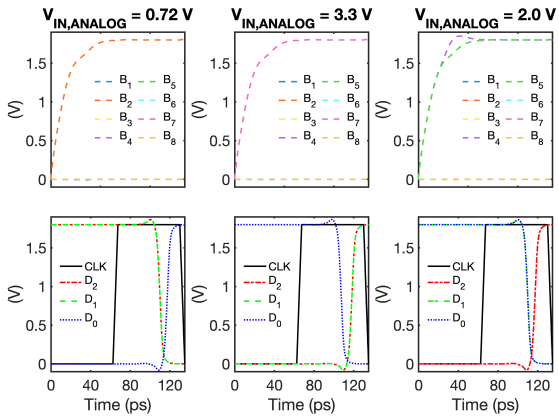


Fig. 9. Transient verification results of the eoADC architecture.

the corresponding MRR generates a dip. For a given input code width (1 LSB of the ADC), only one transmission spectrum produces power lower than the reference, activating a single opto-electric thresholding block (1-hot encoding) as a function of the input voltage.

Fig. 9 shows the transient characteristics of our proposed eoADC for three input settings. The subplots in the top row indicate that for analog inputs of 0.72 V and 3.3 V, only one opto-electric thresholding block, followed by the TIA and amplifier (B_2 and B_7, respectively), is activated. The ROM-based decoder outputs the digital codes 001 and 110, as seen in the bottom subplots. For an analog input of 2 V, two activations (B_4 and B_5) occur since 2 V lies at the boundary between two adjacent codes. However, the ceiling priority ROM-based decoder correctly outputs 100, demonstrating the accuracy, robustness, and reliable operation of our eoADC at a sampling speed of 8 GS/s ($\sim$ 125 ps clock period).

Fig. 10 shows the simulation results of the ADC transfer function and differential non-linearity (DNL). The code width closely matches the ideal, with no missing codes (no DNL of -1 LSB). A 10 µm radius MRR with a 250 nm gap was used to achieve 1-hot encoding, with 200 µW input optical power at 1310.5 nm and 18 µW optical reference power per channel. The ADC operates with 1.8 V analog and digital supply voltages. With a wall-plug efficiency of 0.23 [47], the total optical power is 7.58 mW, and the total electrical power is 11 mW. Eliminating the cascaded amplifiers and

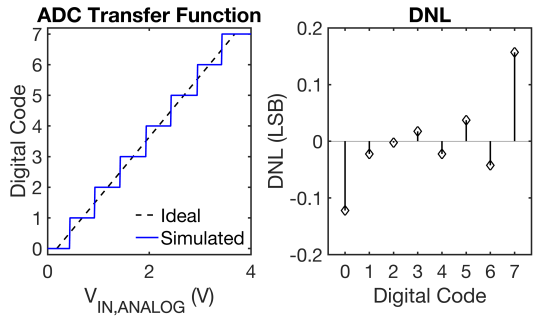


Fig. 10. ADC transfer function (left-subplot) and differential nonlinearity (DNL) characteristics (right-subplot).

TABLE I
PERFORMANCE COMPARISON OF VARIOUS PHOTONIC IMC MACROS.

Reference	Throughput (TOPS)	Power Efficiency (TOPS/W)	Weight Update (Speed)
[33]	0.12	—	60 GHz
[48]	0.93	0.83	$<$ 0.5 GHz*
[49]	11.0	—	2 Hz†
[50]	—	10	$\sim$ 1 GHz ‡
[51]	3.98	1.97	$<$ 0.5 GHz*
This Work	4.10	3.02	20 GHz

* FPGA-controlled multi-channel DC power supply
† Utilizing Finisar WaveShaper 4000S, settling time 500 ms
‡ PCM write speed

TIAs can reduce power consumption but results in slower speed, e.g., the eoADC without these components operates at 416.7 MS/s while consuming 58% less electrical power. In addition, optimizing the MRR's voltage modulation efficiency can improve bit precision and speed.

D. Performance Analysis

A 16×16 photonic tensor core was analyzed to evaluate performance metrics, enabling it to compute 16 vector multiplications of two 1×16 vectors. With 3-bit weight precision, the core incorporates 768 pSRAM bitcells. Four wavelength channels are used for WDM-based multiplication, allowing 4 vector multiplication cores to simultaneously process 1×16 vector multiplications per row. While the MRRs and photodetectors offer high electro-optical bandwidth for fast operations, latency from the electro-optic ADC limits the overall speed. The core achieves a computational throughput of 4.10 TOPS (1 operation = 3-bit multiplication/addition). Power efficiency is calculated considering the pSRAM bitcells, mixed-signal multiplication cores, electro-optic ADC, TIA [52], and a laser wall-plug efficiency of 0.23 [47], yielding 3.02 TOPS/W. Table I compares throughput, power efficiency, and weight update speeds across various photonic IMC macros.

V. CONCLUSION

In summary, we present a novel, mixed-signal, multi-bit, scalable photonic SRAM-augmented tensor core that enables ultra-fast, energy-efficient matrix multiplication computations. The proposed architecture supports multi-GHz memory updates, suitable for large-scale datasets and in-situ training. It leverages fabrication-friendly integrated silicon photonics utilizing GlobalFoundries' monolithic 45SPCLO technology node, enabling seamless integration into existing foundries. Additionally, our novel high-speed, energy-efficient 1-hot encoding electro-optic ADC architecture enables an end-to-end photonic tensor core that is seamlessly compatible with electrical subsystems. Consequently, leveraging the speed, energy efficiency, and bandwidth advantages of photonics, this architecture establishes a robust platform for next-generation, high-performance, and data-centric computing systems.

ACKNOWLEDGMENT

This work is supported by the Defense Advanced Research Projects Agency (DARPA) under Grant No. HR001123S0024.

REFERENCES

[1] W. A. Wulf *et al.*, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[2] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[3] N. Verma *et al.*, "In-memory computing: Advances and prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, 2019.

[4] T. Prodromakis *et al.*, "A review on memristive devices and applications," in *2010 17th IEEE international conference on electronics, circuits and systems*. IEEE, 2010, pp. 934–937.

[5] X.-Y. Xu *et al.*, "Integrated photonic computing beyond the von neumann architecture," *ACS Photonics*, vol. 10, no. 4, pp. 1027–1036, 2023.

[6] L. Gyongyosi *et al.*, "A survey on quantum computing technology," *Computer Science Review*, vol. 31, pp. 51–71, 2019.

[7] C.-J. Jhang *et al.*, "Challenges and trends of sram-based computing-in-memory for ai edge devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1773–1786, 2021.

[8] S. Yin *et al.*, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, 2020.

[9] J. Mu *et al.*, "Sram-based in-memory computing macro featuring voltage-mode accumulator and row-by-row adc for processing neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2412–2422, 2022.

[10] B. Chen *et al.*, "Efficient in-memory computing architecture based on crossbar arrays," in *2015 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2015, pp. 17–5.

[11] H. Li *et al.*, "Four-layer 3d vertical rram integrated with finfet as a versatile computing unit for brain-inspired cognitive information processing," in *2016 VLSI*. IEEE, 2016, pp. 1–2.

[12] R. Khaddam-Aljameh *et al.*, "Hermes core–a 14nm cmos and pcm-based in-memory compute core using an array of 300ps/lsb linearized cco-based adcs and local digital processing," in *2021 Symposium on VLSI Circuits*. IEEE, 2021, pp. 1–2.

[13] M. Le Gallo *et al.*, "A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference," *Nature Electronics*, vol. 6, no. 9, pp. 680–693, 2023.

[14] S. Jung *et al.*, "A crossbar array of magnetoresistive memory devices for in-memory computing," *Nature*, vol. 601, pp. 211–216, 2022.

[15] Y.-C. Chiu *et al.*, "A 22nm 4mb stt-mram data-encrypted near-memory computation macro with a 192gb/s read-and-decryption bandwidth and 25.1-55.1 tops/w 8b mac for ai operations," in *2022 ISSCC*, vol. 65. IEEE, 2022, pp. 178–180.

[16] J.-M. Hung *et al.*, "Challenges and trends indeveloping nonvolatile memory-enabled computing chips for intelligent edge devices," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1444–1453, 2020.

[17] A. Agrawal *et al.*, "Magnetoresistive circuits and systems: Embedded non-volatile memory to crossbar arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 6, pp. 2281–2294, 2021.

[18] K. Roy *et al.*, "In-memory computing in emerging memory technologies for machine learning: An overview," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[19] A. Yusuf *et al.*, "Domain-specific stt-mram-based in-memory computing: A survey," *IEEE Access*, 2024.

[20] K. Cho *et al.*, "Sram write-and performance-assist cells for reducing interconnect resistance effects increased with technology scaling," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 4, pp. 1039–1048, 2022.

[21] P. Oldiges *et al.*, "Chip power-frequency scaling in 10/7nm node," *IEEE Access*, vol. 8, pp. 154 329–154 337, 2020.

[22] M. E. Sinangil *et al.*, "A 7-nm compute-in-memory sram macro supporting multi-bit input, weight and output and achieving 351 tops/w and 372.4 gops," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 1, pp. 188–198, 2020.

[23] Y.-D. Chih *et al.*, "16.4 an 89tops/w and 16.3 tops/mm 2 all-digital sram-based full-precision cim macro in 22nm for machine-learning edge applications," in *2021 ISSCC*, vol. 64. IEEE, 2021, pp. 252–254.

[24] L. M. Shaker *et al.*, "Integrated photonics: bridging the gap between optics and electronics for enhancing information processing," *Journal of Optics*, pp. 1–13, 2023.

[25] C. G. Kibebe *et al.*, "Harnessing optical advantages in computing: a review of current and future trends," *Frontiers in Physics*, vol. 12, p. 1379051, 2024.

[26] P. J. Winzer, "Modulation and multiplexing in optical communications," in *Conference on Lasers and Electro-Optics*. Optica Publishing Group, 2009, p. CTuL3.

[27] B. J. Shastri *et al.*, "Photonics for artificial intelligence and neuromorphic computing," *Nature Photonics*, vol. 15, no. 2, pp. 102–114, 2021.

[28] C. Ríos *et al.*, "In-memory computing on a photonic platform," *Science advances*, vol. 5, no. 2, p. eaau5759, 2019.

[29] S. Ohno *et al.*, "Si microring resonator crossbar array for on-chip inference and training of the optical neural network," *Acs Photonics*, vol. 9, no. 8, pp. 2614–2622, 2022.

[30] J. Feldmann *et al.*, "Parallel convolutional processing using an integrated photonic tensor core," *Nature*, vol. 589, no. 7840, pp. 52–58, 2021.

[31] H. Zhang *et al.*, "On-chip photonic synapses based on slot-ridge waveguides with pcms for in-memory computing," *IEEE Photonics Journal*, vol. 13, no. 2, pp. 1–13, 2021.

[32] Y. Shen *et al.*, "Deep learning with coherent nanophotonic circuits," *Nature photonics*, vol. 11, no. 7, pp. 441–446, 2017.

[33] Z. Lin *et al.*, "120 gops photonic tensor core in thin-film lithium niobate for inference and in situ training," *Nature Communications*, vol. 15, no. 1, p. 9081, 2024.

[34] S. Bandyopadhyay *et al.*, "A photonic deep neural network processor on a single chip with optically accelerated training," in *2023 Conference on Lasers and Electro-Optics (CLEO)*. IEEE, 2023, pp. 1–2.

[35] A. N. Tait *et al.*, "Neuromorphic photonic networks using silicon photonic weight banks," *Scientific reports*, vol. 7, no. 1, p. 7430, 2017.

[36] C. Ríos *et al.*, "Integrated all-photonic non-volatile multi-level memory," *Nature photonics*, vol. 9, no. 11, pp. 725–732, 2015.

[37] J. Sun *et al.*, "A 128 gb/s pam4 silicon microring modulator with integrated thermo-optic resonance tuning," *Journal of Lightwave Technology*, vol. 37, no. 1, pp. 110–115, 2018.

[38] J. Nijem *et al.*, "High-q and high finesse silicon microring resonator," *Optics Express*, vol. 32, no. 5, pp. 7896–7906, 2024.

[39] S. Park *et al.*, "A 4gs/s 4b flash adc in 0.18/spl mu/m cmos," in *2006 IEEE International Solid State Circuits Conference-Digest of Technical Papers*. IEEE, 2006, pp. 2330–2339.

[40] P. C. Scholtens *et al.*, "A 6-b 1.6-gsample/s flash adc in 0.18-/spl mu/m cmos using averaging termination," *IEEE JSSC*, vol. 37, no. 12, pp. 1599–1609, 2002.

[41] S. S. Kumar *et al.*, "A 750mw 24gs/s 12b time-interleaved adc for direct rf sampling in modern wireless systems," in *2023 ISSCC*. IEEE, 2023, pp. 1–3.

[42] M. Guo *et al.*, "Background timing mismatch calibration techniques in high-speed time-interleaved adcs: A tutorial review," *IEEE TCAS-II: Express Briefs*, vol. 69, no. 6, pp. 2564–2569, 2022.

[43] M. Bagheri *et al.*, "Interchannel mismatch calibration techniques for time-interleaved sar adcs," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 420–433, 2021.

[44] M. A.-A. Kaiser *et al.*, "Design of energy-efficient cross-coupled differential photonic-sram (psram) bitcell for high-speed on-chip photonic memory and compute systems," *arXiv preprint arXiv:2503.19544*, 2025.

[45] S. Sunder *et al.*, "Scalable in-memory compute optical processor," in *Smart Photonic and Optoelectronic Integrated Circuits 2025*, vol. 13370. SPIE, 2025, pp. 107–113.

[46] N. Mehta *et al.*, "A laser-forwarded coherent transceiver in 45-nm soi cmos using monolithic microring resonators," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 1096–1107, 2020.

[47] S. A. Blokhin *et al.*, "High power single mode 1300-nm superlattice based vcsel: Impact of the buried tunnel junction diameter on performance," *IEEE Journal of Quantum Electronics*, vol. 58, no. 2, pp. 1–15, 2022.

[48] S. Du *et al.*, "Scalable parallel photonic processing unit for various neural network accelerations," *Photonics Research*, vol. 12, no. 11, pp. 2691–2702, 2024.

[49] X. Xu *et al.*, "11 tops photonic convolutional accelerator for optical neural networks," *Nature*, vol. 589, no. 7840, pp. 44–51, 2021.

[50] W. Zhou *et al.*, "In-memory photonic dot-product engine with electrically programmable weight banks," *Nature Communications*, vol. 14, no. 1, p. 2887, 2023.

[51] H. Ouyang *et al.*, "Computing dimension for a reconfigurable photonic tensor processing core based on silicon photonics," *Optics Express*, vol. 32, no. 18, pp. 31 205–31 219, 2024.

[52] L. Aschei *et al.*, "A 42-ghz tia in 28-nm cmos with less than 1.8% thd for optical coherent receivers," *IEEE Solid-State Circuits Letters*, vol. 3, pp. 238–241, 2020.

An Algorithm-Hardware Co-design Based on Revised Microscaling Format Quantization for Accelerating Large Language Models

Yingbo Hao[†*], Huangxu Chen[‡*], Yi Zou[†§], Yanfeng Yang[†]
[†] *South China University of Technology*, [‡] *Hong Kong University of Science and Technology (GZ)*, Guangzhou, China
[†]{202410193423@mail., zouyi@, mi_yyf@mail.}scut.edu.cn, [‡]{hchen499@connect.}hkust-gz.edu.cn

***Abstract*—The narrow-bit-width data format is crucial for reducing the computation and storage costs of modern deep learning applications, particularly in large language models (LLMs) based applications. Microscaling (MX) format has been proven as a drop-in replacement for the baseline FP32 in existing inference frameworks, with low user friction. However, deploying such a new format into existing hardware systems is still challenging, and the dominant solution for LLM inference at low precision is still low-bit quantization. This particularly limits the strategic applications of such LLMs in real deployment on a large scale. In this work, we propose an algorithm-hardware co-design that adopts a two-level *Revised MX Format Quantization* (RMFQ) and a *Revised MX Format Accelerator* (RMFA) architecture design. RMFQ proposes the *revised MX (RMX)* format and provides a novel quantization framework with innovative group direction. Also, RMFA provides an RMX adaptive hardware architecture and an RMX encoding scheme. As a result, RMFQ pushes the limit of 4-bit and 6-bit quantization to a new state-of-the-art, and RMFA surpasses the existing outlier-aware accelerator such as OliVe, achieving a 1.28× speedup and a 1.31× energy reduction.**

***Index Terms*—Large Language Model (LLM), Revised Microscaling (RMX) Format, Accelerator, Two-Level Quantization.**

I. Introduction

The latest advances in Artificial Intelligence (AI) capabilities, such as conversational question answering and intelligent code completion [1], are primarily achieved by scaling up the underlying deep learning models [2]. However, this scaling has led to a significant increase in the demand for more computational power and memory capacity to keep up with training and deploying such large-scale models.

Meanwhile, model quantization has been proven to be an effective technique for reducing memory usage and computational requirements by representing values with narrower bit widths [3]. Native support for low bit width formats is now prevalent in AI-focused hardware. For example, NVIDIA has introduced INT8 integer type in tensor core since its Turing architecture [4]. Particularly, 8-bit quantization necessitates a scale factor for each tensor and has demonstrated noteworthy impact when the number of model parameters is relatively small [5], [6]. However, when dealing with Transformer-based *Large Language Models* (LLMs), the model's performance is easily affected by a very small portion ($< 0.1\%$) of outliers [7], [8]. For example, regular quantization methods fail at a scale of 6.7B parameters for OPT model [8].

[*]These authors have contributed equally to this work.
[§]Corresponding author.

TABLE I
Quantization Architecture Comparison

Method	Encoding Scheme	Off / On-chip Memory Aligned	Memory Bit Width	Compute Bit Width	Overhead Area Ratio
OLAccel [12]	Coordinate list	No	5.81	4.36	71%
GOBO [11]	Coordinate list	No	4.04/6.81	16	55%
OliVe [10]	Outlier-victim pair	Yes	4.23/8	4.23/8	4%
RMFA (Ours)	RMX detecting	Yes	4.06/6.06	4.06/6.06	2%

Additionally, tensor-level scaling has been shown to be insufficient for Transformer-based LLMs due to their limited dynamic range. Various methods have been proposed to address outliers, with one common approach being to handle normal values and outliers differently [9]–[11]. For example, OliVe [10] adopts an outlier-victim pair (OVP) quantization to sacrifice normal values to accommodate outliers locally. GOBO [11] and OLAccel [12] use coordinate list to distinguish outliers from normal values. However, such progress not only leads to unaligned memory storage and accesses but also causes large area overheads due to the extra outlier controller.

Another approach is to use new data formats to simultaneously accommodate outliers and regular values. Research has shown that using micro-scaled data formats, which associate scaling factors with fine-grained sub-blocks of a tensor, is a promising method to quantize Transformer-based LLMs [13]–[16], and even more effective in the sub-8 bit regime [14]–[16]. However, these efforts are all optimized for existing GPUs, which are architecturally not compatible with these new data formats, often requiring complex dynamic quantization algorithms such as data recognition and reordering. Microscaling (MX) data format [17] has been proposed as the first open standard for a family of micro-scaled data format aimed at deep learning inference. However, the deployment of MX still has a long way to go through the combined efforts of industries and academia.

In this paper, we propose an algorithm-hardware co-design that adopts a two-level *Revised MX Format Quantization* (RMFQ) and a *Revised MX Format Accelerator* (RMFA) architecture design to solve the low-precision quantization problem of LLMs. In the software algorithm, we design a novel quantization framework based on *revised MX* (RMX) format to handle outliers more effectively. At the same time, in the hardware architecture, we design RMFA to support the computation of RMX format, mainly including a systolic

array adapted to RMX format and an RMX encoding scheme. Table I compares RMFQ with previous outlier-aware quantization architecture. RMFA achieves high performance with minimal hardware overhead and can be efficiently integrated into existing hardware accelerators. The main contributions are listed as follows:

- We ***propose*** RMFQ, a two-level quantization method using RMX format that introduces adaptive int/float and a novel group direction. The method significantly reduces the equivalent bit width compared to the original MX format and handles outliers in LLMs more effectively.
- We ***design*** RMFA to benefit from RMFQ, mainly including an RMX adaptive systolic array architecture and a RMX encoding scheme. This architecture optimizes data transfer and storage, thus effectively reducing the bandwidth requirements for data transmission.
- We ***implement*** and ***demonstrate*** the effectiveness of RMFQ on representative LLMs. On one hand, RMFQ achieves state-of-the-art (SOTA) quantization accuracy for LLMs of 4-bit and 6-bit. On the other hand, evaluation of RMFA showcases 3.13×, 1.88×, and 1.28× speedup over the ANT, OLAccel and OliVe, respectively.

II. Background and Proposed Method

In this section, we summarize the challenges in quantizing LLMs and propose the RMX format. Based on this, we raise our novel quantization method.

A. Challenges in Quantizing LLMs

There have been various quantization methods proposed over the past years. The two main classes of methods are post-training quantization (PTQ) [3], [7], [10], [13], [15] and quantization-aware training (QAT) [3], [9], [18]. QAT requires fine-tuning and enables lower bit quantization [19]. However, LLMs retraining is extremely resource-consuming, which makes PTQ a more desirable method for LLMs. Recent state-of-the-art PTQ methods [7], [15] demonstrate that the weights in LLMs can be quantized to four or even three bits via grouping-based (e.g., a scaling factor for g consecutive weights in a tensor) quantization methods with competitive model performance.

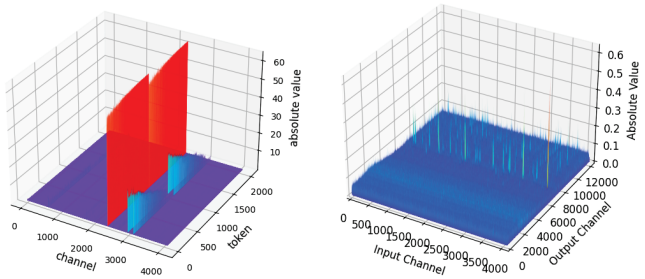
Nevertheless, quantizing activations with the same method doesn't work well due to the existence of outliers. A mainstream approach is to use higher precision for calculations involving outliers [8], [14], [16], but this simultaneously brings complex dynamic quantization perception processes, thus reducing the speed and making system implementation difficult. Fig. 1 visualizes the absolute values of activation and weight tensors of a typical linear layer in OPT-6.7B [20]. It can be seen that outliers only exist in specific channels of activation. The intuitive approach is to use per-channel (e.g., a scaling factor for each channel in an activation tensor) quantization for activations. However, per-channel activation quantization does not map well to hardware-accelerated GEMM kernels. Assume a typical per-channel quantization using INT4. The quantization process can be expressed as:

$$X^{INT4} = Round(\frac{X^{FP16}}{\Delta}), \Delta = \frac{\max(|X^{FP16}|)}{2^{4-1}-1}, \quad (1)$$

where x^{FP16} is the tensor to be quantized, x^{INT4} is the quantized counterpart and Δ is the quantization step size. The matrix multiplication process can be expressed as:

$$Y = \text{diag}(\Delta_x^{FP16}) \cdot (X^{INT4} \cdot W^{INT4}) \cdot \text{diag}(\Delta_w^{FP16}). \quad (2)$$

In those kernels, scaling can only be performed along the outer dimenions of the matrix multiplication (i.e., token dimension of activations). Therefore, previous work all uses per-token or per-tensor activation quantization, even if there is work using a per-channel smoothing factor to keep mathematical equivalence of a linear layer, which is still complicated [13].



(a) (b)

Fig. 1. Absolute values of (a) activation and (b) weight tensors in OPT-6.7B.

B. RMX Format

MX formats have demonstrated their effectiveness on inference of various deep learning benchmarks, including LLMs [21]. However, the approach relies on pseudo-simulation, and current GPU architectures are not compatible with MX format ultimately. Therefore, integrating MX format into existing quantization methods and proposing practical hardware-adaptive architectures is crucial to fully unlocking the potential of MX format. Based on this analysis, we propose the RMX format. Fig. 2 compares MX format with our proposed RMX format.

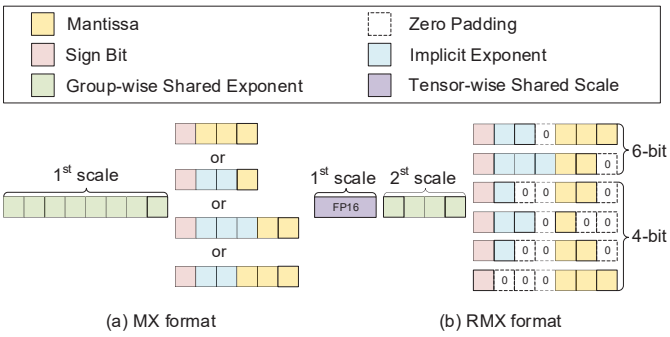


Fig. 2. Comparison between MX and RMX format

1) Mixed Types and Precisions: Firstly, we replace the fixed integer (e.g., MXINT4) or floating-point (e.g., MXFP4, MXFP6) number with a mixed int/float type and 4-bit/6-bit precision format. This is based on the observation that in different layers of a neural network, the distribution characteristics of each tensor are different, possibly being uniform-like, Gaussian-like, or Laplace-like [9]. It is not appropriate to use the same data type for different layers, such as using MXINT4

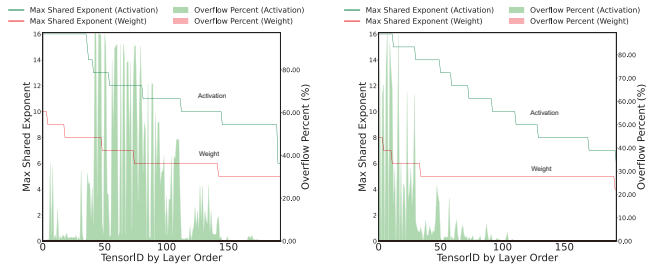
for activation tensors. Moreover, the modification is hardware-friendly. All these 6 data types can be implemented in a unit supporting E3M3 (3 implicit exponent, 3 mantissa) for *multiplication and accumulation* (MAC) operations. Note that zero padding is used to illustrate the computing mechanism of RMX format; 4/6-bit RMX format remains its original bit width in memory storage and access.

2) Reduced Equivalent Bit Width bit_e: Secondly, we replace the one-level group-wise scale factor with the two-level tensor-wise plus group-wise scale factor. Since both MX and RMX formats are micro-scaled formats that have small granularities, the group-wise scale factors have a considerable memory overhead. To this end, we propose a new metric denoted as bit_e to calculate the equivalent bit width of micro-scaled formats and measure the memory-saving effects. bit_e is defined as:

$$bit_e = \frac{bit \cdot g + bit_s}{g}, \tag{3}$$

where bit is the quantization bit width, g is the group size and bit_s is the bit width of the group-wise scale factor. Assume g = 64, which we will illustrate in Section IV-B.

For example, in 4-bit quantization, assume an activation tensor of [4096, 2048] for OPT-6.7B on WikiText [22]. The average bit width burden of the tensor-wise scale factor on each group is $16 \div \frac{4096 \cdot 2048}{64} = 10^{-4}$, which is too small to be considered in bit_e. Therefore, bit_s is crucial to bit_e. As shown in Fig. 3, we calculate the max shared exponent and shared exponent overflow ($> 2^3 - 1 = 7$) for OPT-6.7B and GPT2-XL [1]. It is evident that the shared exponent of the weights rarely experiences overflow. However, there is a significant amount of overflow in the activations. Therefore, we choose 4 as our group-wise shared exponent bit width for a max shared exponent of 15. As a result, bit_e for the original MX format is $\frac{4 \cdot 32 + 8}{32} = 4.25$, and bit_e for the proposed RMX format is $\frac{4 \cdot 64 + 4}{64} = 4.06$.



(a) OPT-6.7B on WikiText (b) GPT2-XL on WikiText

Fig. 3. Max shared exponent and shared exponent overflow percent for representative LLMs. The overflow percent for weights is too small, thus hard to see in the figure.

C. Quantization Method

Our intuition is to leverage the implicit scale factor in the RMX format, enabling group-wise quantization to be performed along the inter dimensions of matrix multiplication. Fig. 4 illustrates the proposed two-level quantization method. We use the typical per-tensor method for 1st quantization. To handle the channel-wise outliers of activation tensors, we revise the previous group direction of activation tensors; instead, we group along the channel direction. For weight tensors, we find the same pattern of distribution compared to activation tensors, which is shown in Fig. 1. Therefore, we group along the output channel direction to handle the less significant outliers in weight tensors too. Nonetheless, we propose the hardware system that is compatible with such group direction in Section III. Therefore, the 1st 4-bit quantization process can be expressed as:

$$\overline{X}^{FP16} = \frac{X^{FP16}}{\Delta^{1st}}, \Delta^{1st} = \frac{\max(|x^{FP16}|)}{\max(|Q^{grid}|)}, \tag{4}$$

where Q^{grid} is the quantization grid (e.g., [-8,..., 6, 7] for RMXINT4), $\overline{X}^{FP16}$ is the 1st quantization result. The 2nd quantization process for a group can be expressed as:

$$\frac{1}{\Delta^{2nd}} = \max(\lfloor \log_2 \frac{\max(|Q^{grid}|)}{\max(|\overline{X}^{FP16}_{group}|)} \rfloor - 1, 15), \tag{5}$$

$$X^{RMX4} = Round(\frac{\overline{X}^{FP16}_{group}}{\Delta^{2nd}}), \tag{6}$$

where $\overline{X}^{FP16}_{group}$ is the 1st quantization result of a group, $\lfloor \cdot \rfloor$ is the rounding-to-floor function and X^{RMX4} is the 2nd quantization result. We can obtain the optimal group-wise shared exponent by performing the above quantization process.

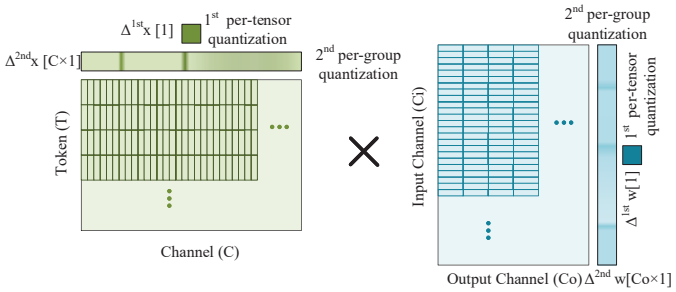


Fig. 4. Illustration of the proposed two-level revised MX format quantization, i.e., RMFQ, method.

We also adopt *Mean Square Error* (MSE) as the quantization optimization target, which has been shown to be effective by many previous studies [3], [9], [23]. The MSE metric can be defined by the following equation:

$$MSE = \int (X^{FP16} - X^{RMX4})^2 \, p(x) \, dx,$$

where $p(x)$ is the probability density function. By searching the minimum MSE, we can select the optimal RMX data types for each layer and quantize layers with the greatest MSEs with higher precision, that is, do mixed data types/precisions quantization.

III. Hardware Architecture

In this section, we describe in detail the contributions from this work in the aspects of RMFA, mainly including an improved systolic array and a hardware-friendly RMX encoding scheme.

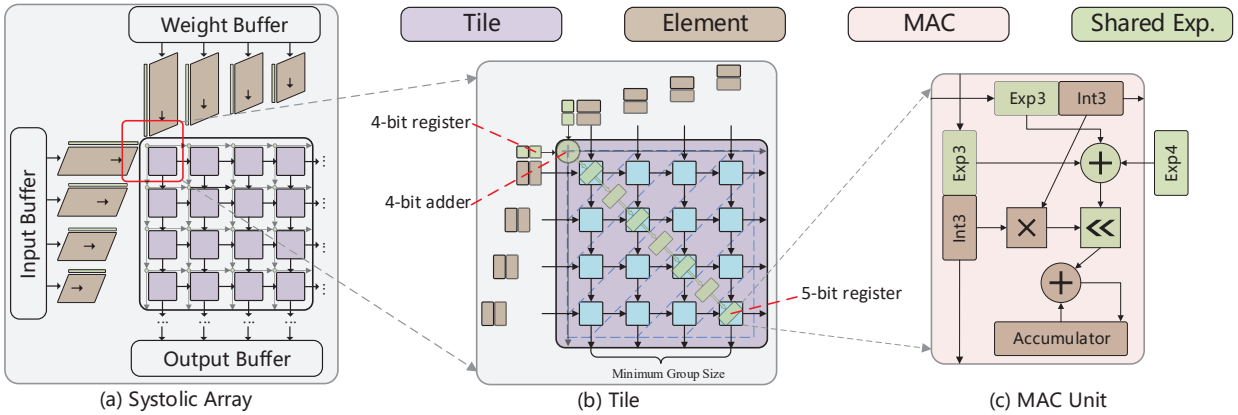


Fig. 5. A brief overview of the systolic array architecture for RMFA.

A. Systolic Array Architecture for RMFA

Fig. 5(a) presents the proposed systolic array architecture. Given that the group size may vary from the array size, it becomes necessary to segment the systolic array into tiles of the smallest viable group size. Each tile computes one group independently per cycle, with the shared exponent being propagated in a systolic fashion. It is evident that a reduction in group size leads to an increase in the number of tiles, thereby incurring more hardware overhead for a given array size. For instance, with an array size of $a = 16$ and a group size of $g = 4$, the additional hardware requirements include $(\frac{a}{g})^2 \times (2g-1) = 112$ 5-bit registers, $2(\frac{a}{g})^2 = 32$ 4-bit registers, and $(\frac{a}{g})^2 = 16$ 4-bit adders. Nonetheless, while decreasing the group size may enhance precision, it also significantly increases both hardware and bandwidth overheads.

Fig. 5(b) depicts a tile designed for the RMX format with an example group size of $g = 4$. Due to the intrinsic characteristics of the systolic array, the next columns of data must be transmitted to the next column of MAC units one cycle later. Consequently, data belonging to the same group is initially partitioned through a grid, with corresponding shared exponents denoted by identical colors. The matrix operation formula for the MX data type is expressed as:

$$\sum_{1}^{n}\sum_{1}^{g} 2^{e_s^a} a_{mx} \times 2^{e_s^w} w_{mx} = \sum_{1}^{n}\sum_{1}^{g} (a_{mx} \times w_{mx}) \ll e_s^a + e_s^w,$$

where $\ll$ means bit shifting operation, e_s^a and e_s^w are the shared exponents for activation and weight respectively, a_{mx} and w_{mx} are activations and weights in either RMX4 or RMX6, n and g are group number and group size. Therefore, we first sum the shared exponents of the activations and weights, followed by shifting the resultant product.

The modified MAC unit is illustrated in Fig. 5(c). In contrast to the conventional MAC unit, the proposed model necessitates the substitution of the dual-input shifter with a triple-input shifter and an expansion in the accumulator's width to accommodate the shift range pressure induced by the shared exponents.

B. Hardware-friendly RMX Encoding Scheme

The computation of the shared exponent is pivotal in the RMFA. Once determined, the encoding and decoding algorithms can be straightforwardly applied based on the RMX format specification. The quantization circuit is depicted in Fig. 6. Data resulting from nonlinear operations, typically in FP16 or BF16 format, is initially scaled by a predetermined factor from the software calibration phase and truncated to a 16-bit integer. This truncation preserves precision for groups with smaller magnitudes. For each data group, the circuit identifies the position of the first '1' within the top 16 bits, encoding this position as a 4-bit integer to ascertain the group's shared exponent. Subsequently, the data is right-shifted, truncated to 8 bits, and quantized to RMX data according to the specified RMX type.

Algorithm 1 RMX Encoding Algorithm

Input: $\textbf{Group_Data}_{G,N}$, **Scale**, **RMX_Type**
Output: $\textbf{Group_Data}_{G,N}$, $\textbf{Shared_Exponent}_N$

$Group_Data_{G,N} \leftarrow Group_Data_{G,N} * Scale$
$Group_Max_Data_N \leftarrow Max(Group_Data_{G,N}, dim = 0)$
$Max_Data \leftarrow Max(Group_Max_Data_N)$
$Shared_Exponent_N \leftarrow Frexp(\frac{Max_Data}{Group_Max_Data_N})$
$Group_Data_{G,N} \leftarrow Group_Data_{G,N} * (2^{Shared_Exponent_N})$
$Group_Data_{G,N} \leftarrow Quantize(Group_Data_{G,N}, RMX_Type)$
return $\text{Group_Data}_{G,N}$, Shared_Exponent_N

At the software level, the primary goal is to simulate the error introduced by the quantization algorithm. The algorithm 1 outlines the RMX encoding process. Initially, group data undergoes scaling. Subsequently, the sharing index is determined by comparing the maximum value within the group to the global maximum value, utilizing the $Frexp$ function to extract the exponent of the floating-point number. Finally, all data are quantized to integers, with shifts applied based on the shared index, to approximate the error induced by tensor quantization.

During hardware inference, the quantization circuit is depicted in Fig. 6. Data resulting from nonlinear operations, typically in FP16 or BF16 format, is initially scaled by a predetermined factor from the software calibration phase and truncated to a 16-bit integer. This truncation preserves precision for groups with smaller magnitudes. For each data group, the circuit identifies the position of the first '1' within the top 16 bits, encoding this position as a 4-bit integer to ascertain the group's shared exponent. Subsequently, the data

is right-shifted, truncated to 8 bits, and quantized to RMX data according to the specified RMX_type.

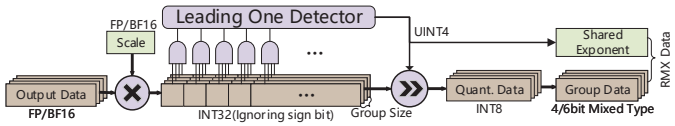


Fig. 6. The Hardware Architecture of the RMX Encoder

IV. Experiment

A. Methodology

Quantization Framework and Benchmark To evaluate our proposed RMFQ framework, we deploy our experiments on PyTorch. To demonstrate the general applicability of our proposed method on LLMs, we evaluate our method on the OPT suite [20], LLaMA [24], Llama-2 [25] with varying model sizes ranging from 6.7B to 13B, and GPT2-XL [1]. Outliers become a significant problem when model size scales up to about 6.7B. We evaluate language modeling tasks using the WikiText (Wiki) [22] and C4 [26] datasets. We use perplexity as an evaluation metric, which is a widely used one for autoregressive models; lower perplexity means better model performance. Additionally, to prove the generality of our method, we conduct evaluations on ResNet-18 [27] using the ImageNet dataset [28]; higher accuracy means better model performance.

Quantization Baselines We compare our framework with existing quantization frameworks, including ANT [9], OliVe [10], and Tender [29]. ANT is a quantization framework that introduces a novel data type to better represent data and adapt to different data formats between layers. OliVe is a hardware-friendly quantization framework that uses the OVP encoding method to protect outlier values while ensuring memory alignment. Tender is a decomposed quantization technique that uses powers of two to represent scale factors of decomposed matrices to avoid explicit requantization. We conduct PTQ experiments for evaluation. Each the PTQ quantization uses 128 randomly sampled calibration datasets.

Accelerator Baselines We compare the performance and energy of the RMFQ accelerator with three hardware-friendly quantization accelerators, including OliVe, ANT, and OLAccel. We use 4-bit and 6-bit mixed precision quantization to ensure the same perplexity as the other three quantization frameworks.

RMFQ Implementation Our evaluation of the proposed systolic array and encoder is conducted using the Synopsys Design Compiler (DC) tool [30]. We opt for the TSMC N28 technology node, setting the timing constraint to a relaxed $2ns$ to facilitate a fair comparison in terms of area and power consumption metrics. Furthermore, we standardize both the systolic array and group sizes to 64 for consistency in our analysis. Additionally, we use CACTI [31] to evaluate the area and power consumption of the buffer. We perform performance and energy simulations using a cycle-level simulator, DnnWeaver [32]. We use DeepScaleTool [33] to scale the other designs to the same technology node for a fair comparison.

B. Quantization Performance Results

Group Size Exploration We utilize both CNN models and Transformer-based LLMs for computer vision and natural language processing tasks. Experimental results for 4-bit RMX on WikiText are shown in Fig. 7. It is evident that the group size ranging from 8 to 64 has negligible impact on models' accuracy, and the effect on 6-bit RMX can only be less significant. However, as the group size continues to increase, the impact on the model's accuracy becomes significant.

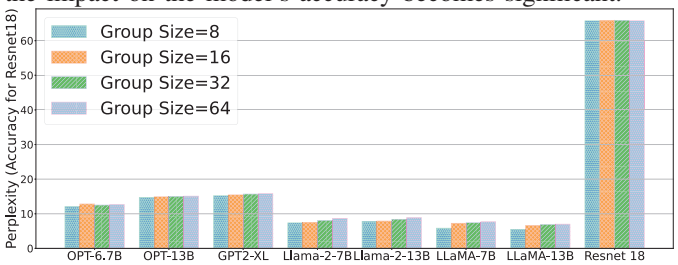


Fig. 7. Accuracy Comparison of Different Group Size

To further demonstrate the superiority of RMFQ, we conduct experiments on the data types and group directions using the OPT-6.7B model on WikiText. The experiment result is shown in Fig. 8. We only show the quantization method for activation because the quantization method for weight is paired. It can be seen that token-wise group quantization that we adopt demonstrates greater robustness with increasing group size, and the 4-bit mixed RMX format outperforms RMXINT4. Additionally, channel-wise group quantization also achieves competitive results with small group sizes, indicating the potential of RMFQ integration in existing GEMM kernels (e.g., tensor cores).

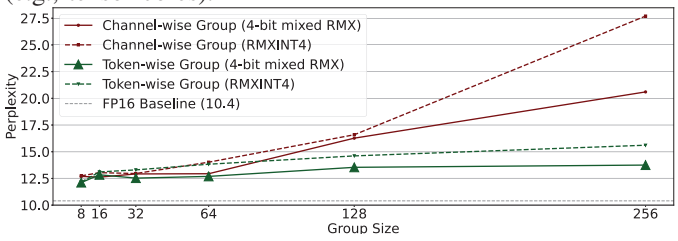


Fig. 8. Accuracy Comparison under Different Circumstances

Benchmark and Comparison For 4-bit RMFQ, we benchmark against INT8, while for 6-bit RMFQ, we benchmark against FP16. The results are presented in Table II. To summarize, the proposed RMFQ framework sets a new SOTA for 4-bit and 6-bit quantization of LLMs. In all LLM tasks, 4-bit RMFQ achieves performance surpassing INT8, while 6-bit RMFQ is almost lossless (maximum perplexity increase of 3.4% compared to the FP16 baseline). Compared to 6-bit quantization, the impact of outliers on model performance is significantly greater in 4-bit quantization. This is due to the fact that quantizing outliers with others results in a larger scale factor. This effect becomes more pronounced at the inherently lower quantization levels of 4-bit PTQ. Our proposed RMFQ allows outliers and normal values to be quantized in isolated groups and provides smaller quantization granularities. Across all scenarios, our approach surpasses existing works, including ANT, OliVe, and Tender. 4-bit RMFQ shows far better

TABLE II
PERFORMANCE METRICS OF VARIOUS MODELS AND CONFIGURATIONS

Method	OPT-6.7B		OPT-13B		GPT2-XL		Llama-2-7B		Llama-2-13B		LLaMA-7B		LLaMA-13B		Resnet-18	
	Wiki ↓	C4 ↓	Wiki	C4	Wiki	C4	Wiki	C4	Wiki	C4	Wiki	C4	Wiki	C4	PTQ ↑	QAT ↑
INT8	16.91	17.28	4E+3	4E+3	15.51	17.35	8.79	9.96	20.52	22.23	8.52	9.73	7.49	8.61	69.38	69.64
4-bit ANT	4E+4	4E+4	4E+3	5E+3	25.39	29.57	142.73	161.34	113.27	119.32	60.56	66.71	73.75	80.32	59.52	65.67
4-bit OliVe	43.44	32.41	45.61	37.56	16.36	18.14	34.24	38.85	36.73	40.12	28.64	30.83	24.12	25.77	59.49	65.61
4-bit Tender	13.56	13.71	16.43	17.33	16.12	17.93	36.47	41.72	55.08	28.14	23.85	25.26	13.68	14.11	64.23	66.41
4-bit RMFQ*	**12.69**	**12.30**	**15.11**	**15.96**	**15.83**	**17.48**	**7.39**	**8.25**	**7.83**	**8.97**	**6.95**	**7.91**	**6.56**	**7.62**	**65.74**	**67.79**
FP16	10.40	10.63	10.13	10.32	14.69	16.30	5.83	6.36	5.25	6.43	5.86	6.97	5.19	5.49	69.68	69.87
6-bit ANT	11.76	11.90	11.42	11.31	14.84	16.46	6.23	6.91	6.03	7.29	6.82	7.93	6.71	6.93	68.99	69.54
8-bit OliVe	10.93	10.73	10.28	10.47	**14.70**	16.37	6.16	6.62	5.98	7.01	6.79	7.52	6.23	7.03	68.82	69.45
8-bit Tender	10.93	10.76	**10.17**	10.39	14.72	16.35	6.09	6.53	5.43	6.61	6.05	7.24	5.41	5.58	69.21	69.89
6-bit RMFQ	**10.48**	**10.67**	10.21	**10.38**	14.72	**16.33**	**6.03**	**6.47**	**5.39**	**6.57**	**6.03**	**7.12**	**5.34**	**5.52**	**69.30**	**70.02**

* Presenting best results using a group size of 8 for best 4-bit quantization method exploration, which is close to results with a group size of 64. Note that RMFQ still outperforms all other methods with a group size of 64.

perplexity than others, especially in Llama-2 and LLaMA. 6-bit RMFQ results are comparable to those of 8-bit OliVe and Tender and surpass them in most situations.

C. Accelerator Implementation Results

Area and Power Table III presents the architectural configuration of MXFA. At an operating frequency of 500 MHz, MXFA occupies an area of 3.09 mm^2 and consumes 472.5 mW of power. The systolic array integrates the RMX format decoding circuitry as described by Section II-B. To ensure sufficient computational throughput, we design a 64 KB input buffer, a 32 KB weight buffer, and a 32 KB output buffer, with the input buffer having double the capacity to facilitate ping-pong operations. Additionally, we standardize the number of MAC units and buffer sizes across all baseline designs. For a fair comparison, we set the accumulator bit width of all MAC unit designs to 24 bits for subsequent performance and energy evaluations.

Performance Fig. 9(a) shows the normalized cycles of different accelerator designs across various models. ANT exhibits the worst performance due to the lack of protection for outliers. Although OLAccel employs mixed precision to protect outliers, its performance is hindered by unaligned memory access and bandwidth limitations. OliVe uses OVP encoding to protect outliers with negligible bandwidth overhead, but this sacrifices accuracy, leading to poor performance when trying to restore the same accuracy. RMFQ achieves better accuracy with negligible bandwidth overhead by sharing exponents, resulting in superior performance at the same accuracy level. Overall, in LLMs, RMFA achieves performance improvements of 3.13×, 1.88×, and 1.28× compared to ANT, OLAccel and OliVe, respectively.

Energy Fig. 9(b) shows the static and dynamic energy consumption of different accelerator designs across various models, including DRAM, on-chip buffers, and cores. Due to the hardware-friendly systolic array design of RMFA, it exhibits the lowest energy consumption. In comparison, ANT and OLAccel consume the most energy due to mixed precision, which resulted in more memory accesses and computation cycles. Compared to OliVe, the sacrifice of accuracy leads to more high-precision layers and higher energy consumption. Overall, in LLMs, RMFA achieves energy reductions of 2.38×, 1.86×, and 1.31× compared to ANT, OLAccel and OliVe, respectively.

TABLE III
AREA AND POWER OF RMFA

Component	Setup	Area/mm^2	Power/mW
Systolic Array*	64×64 MACs	1.41	249.9
Encoder	64	0.03	3.0
Input Buffer	64KB	0.49	49.5
Weight Buffer	32KB	0.65	94.6
Output Buffer	32KB	0.51	75.5
Total		3.09	472.5

* Decoding circuits of RMX format is included in the systolic array.

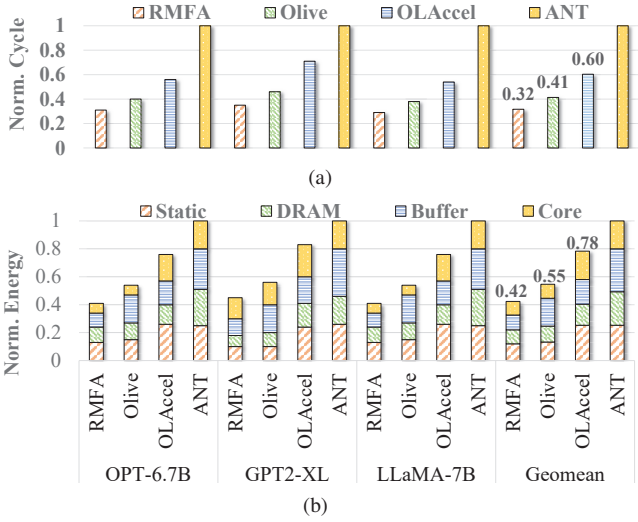


Fig. 9. Comparison of the normalized (a) cycle and (b) energy different accelerator designs

V. CONCLUSIONS

In this paper, we propose an algorithm-hardware co-design for accelerating LLMs. To reduce the quantized equivalent bit width and allow for mixed data types/precision quantization, we propose the RMX format. To handle outliers in LLMs more effectively, we propose the two-level RMFQ with novel group direction. Furthermore, we design RMFA to efficiently support the RMX format. In conclusion, by integrating the RMFQ for low-bit quantization like 4/6-bit, RMFA realizes significantly 1.28× performance and 1.31× energy efficiency improvement in LLMs compared to the prior design such as OliVe with negligible accuracy loss.

VI. ACKNOWLEDGMENT

This research is supported partly by SBSTR Fund No. KJZD2024090310270801, as well as partly by the SCUT Fund No. K3200890.

REFERENCES

[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[4] J. Burgess, "Rtx on—the nvidia turing gpu," *IEEE Micro*, vol. 40, no. 2, pp. 36–44, 2020.

[5] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," in *International conference on machine learning*. PMLR, 2021, pp. 5506–5518.

[6] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-bert: Hessian based ultra low precision quantization of bert," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8815–8821.

[7] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024.

[8] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 318–30 332, 2022.

[9] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

[10] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[11] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.

[12] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 688–698.

[13] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[14] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[15] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, "Qserve: W4a8kv4 quantization and system co-design for efficient llm serving," *arXiv preprint arXiv:2405.04532*, 2024.

[16] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate llm serving," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 196–209, 2024.

[17] B. Darvish Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar *et al.*, "With shared microexponents, a little shifting goes a long way," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[18] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8bert: Quantized 8bit bert," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 2019, pp. 36–39.

[19] K. Chowdhary and K. Chowdhary, "Natural language processing," *Fundamentals of artificial intelligence*, pp. 603–649, 2020.

[20] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[21] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf *et al.*, "Microscaling data formats for deep learning," *arXiv preprint arXiv:2310.10537*, 2023.

[22] W. contributors, "68-95-99.7 rule - wikipedia," in *The Free Encyclopedia*. [Online], 2022.

[23] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[24] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[25] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[26] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[29] J. Lee, W. Lee, and J. Sim, "Tender: Accelerating large language models via tensor decomposition and runtime requantization," *arXiv preprint arXiv:2406.12930*, 2024.

[30] P. Kurup and T. Abbasi, *Logic Synthesis Using Synopsys*, 2nd ed. Springer Publishing Company, Incorporated, 2011.

[31] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Cacti 6.0: A tool to model large caches. hp laboratories," *Tech. Rep. HPL-2009-85*, 2009.

[32] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas." in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Ieee, 2016, pp. 1–12.

[33] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era." in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. Ieee, 2021, pp. 1–5.

LA-MTL: Latency-Aware Automated Multi-Task Learning

Shambhavi Balamuthu Sampath[1,2†] Sami Sawani[1†] Moritz Thoma[1,2] Lukas Frickenstein[2]
Pierpaolo Mori[2,3] Nael Fasfous[2] Manoj Rohit Vemparala[2] Alexander Frickenstein[2]
Ulf Schlichtmann[1] Claudio Passerone[3] Walter Stechele[1]

† Equal Contribution [1]Technical University of Munich [2]BMW Group [3]Politecnico Di Torino

***Abstract*—Multi-Task Learning (MTL) aims to unify a variety of tasks into a single network for improved training and inference efficiency. This is particularly attractive for real-time applications that require simultaneous execution of multiple workloads in resource-constrained embedded environments. However, most MTL approaches focus on enhancing parameters efficiency and overall tasks metrics, often lacking explicit inference latency awareness in the optimization loop. The design space exploration should not compromise on the parameters efficiency or task accuracy objectives in order to meet latency requirements. To address this, we propose LA-MTL, an automated layer-level MTL policy search that incorporates a novel analytical latency factor (ALF). By accounting for local and global latencies during the MTL policy search, we derive solutions that balance task metrics, parameters efficiency and latency constraints. LA-MTL search on ResNet34 yields solutions with up to 50% lower latency on the Jetson AGX Orin while maintaining competitive metrics in semantic segmentation and depth estimation tasks with a +/-2 p.p., on the CityScapes dataset. Additionally, we achieve a superior parameters efficiency, surpassing the state-of-the-art MTL parameters reduction by over 20 p.p. Experiments on benchmark datasets (CityScapes, NYUv2) demonstrate the effectiveness of our approach across various backbones including ResNet34, MobileNetV2, and MobileOne in its expanded form. Code is available at https://github.com/shamvbs/LA-MTL.** [1]

***Index Terms*—multitask learning, convolutional neural networks (CNNs), edge devices, gradients conflict, tasks interference, proxy latency.**

I. Introduction

As the number of tasks and model size increases, memory footprint and computational complexity also grow, making deployment on resource-constrained devices increasingly difficult. This challenge has driven the adoption of multitask learning (MTL), where a single model is trained to perform multiple tasks simultaneously. By sharing layers across tasks, classic MTL offers significant parameters reduction without sacrificing accuracy. However, these benefits can be insufficient for latency-sensitive applications such as autonomous driving, where rapid processing is crucial. A vehicle must detect and locate objects, estimate distances, and perform other functions in real-time. The effect of latency reduction on parameters efficiency and task accuracy is often hard to predict, since they are not directly proportional to each other. In our quest to incorporate latency awareness into MTL, we ask the following questions. *(1) Is there a positive synergistic relationship between latency optimization and classic MTL goals, and can we leverage this? (2) If tasks accuracy or parameters efficiency is sacrificed for faster inference, can these losses be recovered?* The answer to these questions may very well change for different backbones and/or tasks. Therefore, there is a need for a latency-aware optimization scheme that is able to reflect latency penalty for various configurations. At the same time, dealing with these concerns can be cumbersome and time-consuming if not conveniently automated as a part of one cohesive latency-aware MTL framework. To address these challenges, we introduce LA-MTL, with the following contributions:

- ***Analytical Latency Factor:*** A novel latency proxy metric, capturing both local and global context of the multi-task model using a unitless mapping cost at the layer level.
- A ***latency loss*** to adhere to the latency constraints, that respects a user-defined analytical latency requirement based on the target latency.
- ***Gradients/capacity modification mechanism*** for modifying gradients and capacity of shared layers across different tasks to minimize task interference and improve overall task performance.

II. Related Works

a) MTL for Dense Predictions: Various strategies have been used to build MTL solutions. Manual adjustments, which rely on domain expertise, often prove suboptimal due to the difficulty in identifying the optimal configuration within a typically large search space [1] [2]. Another approach is policy learning through Neural Architecture Search (NAS), which involves determining specific neural network depth and width as policy decisions. While NAS introduces a degree of automation, its implementation can vary; some approaches restrict the backbone capacity [3], while others are more adaptive [4], facilitating trade-offs between performance and resource usage. Following [4], our framework, LA-MTL allows the MTL model capacity to dynamically change based on the tasks needs.

b) Hardware-Aware Neural Architecture Search (HW-aware NAS): Hardware-awareness in vision MTL has often been overlooked. Nevertheless, the concept of HW-aware NAS has been extensively explored outside of vision MTL to optimize for device-specific constraints. Proxy-based latency-aware NAS methods typically suffer from poor generaliza-

[1]Correspondence to: shambhavi.balamuthu-sampath@tum.de

tion across diverse hardware architectures [5]–[7]. Some approaches directly incorporate real hardware metrics such as latency into the NAS process [8]–[12]. Additionally, lookup tables (LUTs) are pre-built in certain methods to capture hardware measurements, which can reduce the need for costly, real-time hardware-in-the-loop setups [13]. Even the vision MTL contributions that incorporate latency awareness [14] [15], often do so while being tightly coupled to a specific architectures, HW, and design choices- which necessitates significant manual fine-tuning efforts to extend to other backbones and design space encodings. LA-MTL, on the other hand, requires minimal changes to work with, and supports any convolutional architecture to be optimized for edge deployment.

c) Tasks Interference and Gradients Conflict: An orthogonal MTL strategy involves viewing MTL as a multi-objective optimization problem aimed at minimizing task interference [16]. Effective parameters sharing can induce positive inductive biases between tasks, but excessive sharing can cause gradients conflict. In gradient descent algorithms, the optimization of one task gradients can inadvertently affect the other tasks' gradients moving them into a worse solution point in the design space. Many works modify gradients magnitude and direction to alleviate the problem [17] [18] [16].

Often, MTL training pipeline involves several stages. Interference does not occur during every optimization step. Therefore, it is essential not only to manage task conflicts but also to identify when they occur. Interference is also influenced by the model architecture used in the search space. Our LA-MTL approach facilitates the handling of gradients at different training stages, thereby improving the management of MTL conflicts for a given setup.

Existing SOTA methods that are most relevant to our approach are presented in table I. AutoMTL [4] supports any convolutional backbone with minimal effort but lacks gradient conflict handling and latency-awareness. muNet [19] eliminates conflicts by updating parameters for one task at a time but is not latency-aware. MGD [16] handles interference by deriving Pareto optimal solutions but requires manual integration and is not latency-aware. MTNAS [20] includes latency-awareness but lacks conflict handling and requires adaptation for different backbones and devices. In contrast, our method supports any convolutional backbone with minimal modifications, automatically handles operator-level gradient conflicts, and incorporates latency-awareness. This combination of features ensures efficient and effective multi-task learning across different setups, setting our method apart from current SOTA approaches.

Related Work	Gradients Conflict Handling	CNN-Backbone-Agnostic	Latency Awareness
Auto-MTL [4]	✗	✓	✗
muNet [19]	✓	✓	✗
MGD [16]	✓	✗	✗
MTNAS [20]	✗	✗	✓
LA-MTL (Ours)	✓	✓	✓

TABLE I: Overview of Related Works vs. LA-MTL.

III. LA-MTL Framework

We design and implement a Latency-Aware MTL solution to ensure that the inference latency constraints are met while maintaining competitive task metrics performance. With every layer in a given backbone containing operations like convolution or batch normalization, we follow the operator-level search space encoding from AutoMTL [4]. For the set of N tasks denoted as $\mathbf{T} = \{t_1, t_2, ..., t_N\}$, each task t_i can either share the original backbone operator, has its own copy, or skip that layer's operator (unity or down-sampling). As shown in Fig. 1, such an encoding provides diverse and flexible modulations of a given backbone.

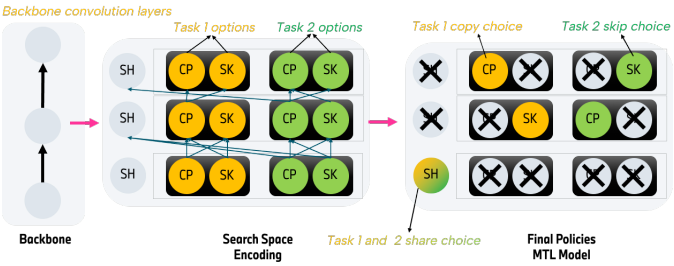


Fig. 1: AutoMTL [4] exponential search space with operator-level encoding. The differentiable search algorithm learns a Gumbel-Softmax distribution and samples an MTL model with fixed branching decisions.

We search for good branching decisions, by learning a branching policy distribution to arrive at a parameters efficient, low latency, and competitive accuracy solution. The overall pipeline is illustrated in Fig. 2. The training pipeline is split into three phases: In the first warm-up phase, no policy training is involved. Rather, we feed forward the average of the original operator, copied operator, and skip connection. The second stage jointly optimizes the sharing policy and the model parameters. Then, a multi-task model is generated by sampling from the learned policy distribution. A final third training stage is performed where the model parameters are trained from scratch while the branching policy is fixed.

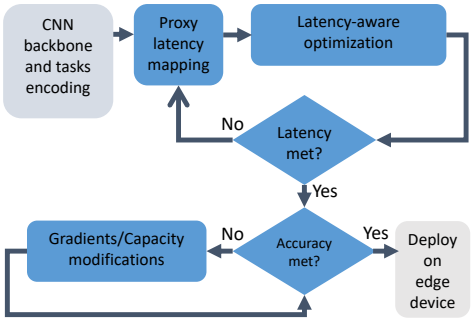


Fig. 2: The main components of our work: We encode our proxy ALF at the operator level. Then, we optimize latency-aware loss with optional gradients/ capacity adjustments.

A. Analytical Latency Factor: A Proxy Latency

Proxy metrics like FLOPs can be used to approximate actual latency, but they do not always correlate well. A reduction in FLOPs can also lead to an increase in latency instead of a decrease, depending on the model's architecture and hardware constraints. Yet, when working well, proxies remain attractive due to their minimum overhead during MTL search space exploration/optimization. To address this, we propose the *Analytical Latency Factor* (ALF), a flexible, unitless proxy metric for latency. For all backbone architectures and task configurations, ALF mapping is adjusted accordingly. The ALF is initially applied uniformly across all layers as follows:

- Original Convolution: To have a lower latency than the single-task approach using the given backbone, we model the original convolution operator as having maximum mapping cost with an ALF factor of 1.
- Task Copy: Since it is the same operation as the original layer but with independent parameters, we also assign it an ALF of 1.
- Skip Connection: A skip connection that applies a unity operation analytically costs 0. If the skip connection down-samples, the mapping becomes of 0.5.

The initial mapping above is obtained from empirical latency profiling of individual layers' forward pass. We recognize that this coarse-grained approach may not always accurately reflect the true computational/latency cost, especially in cases where the choice of operations introduces complex inter-dependencies between layers. To address this, we allow granular control over the latency factor mapping, particularly for cases where skipping a layer can reduce the computational load for that layer but may increase the size of the tensor passed to subsequent layers, potentially leading to orders of magnitude more FLOPs and increased latency. We define down-sampled convolutions as 1x1 convolution kernels of the original backbone operator. For example, we retain the original depthwise or dilated convolutions but down-sample them, unlike [4] that always forces to standard 1x1 convolution while down-sampling. The ALF is implemented in two main contexts:

- *Local ALF*: this factor sums up the latency costs associated with each operator/layer in a given task's path.
- *Global ALF*: this factor captures the interactions between multiple tasks. The global ALF considers the combined inference paths of all tasks, encouraging configurations that enable parallel task execution.

The global ALF does not interfere with the local optimality of each task's analytical latency, but rather, it encourages simultaneous activation of task paths for a given hardware that is able to leverage such inference paths. Fig. 3 illustrates local/ global ALF concept with an example calculation.

B. Hardware-Aware Loss

For $\mathcal{N}$ tasks and $\mathcal{L}$ layers, The chosen policy choice for task i will be denoted by $\mathcal{P}_l^{t_i}$. This is a simple discrete variable that is 0 if the backbone operator is chosen, 1 if the task-specific copy is chosen, or 2 for skip connection. The Gumbel-Softmax approximation is denoted by the below policy variable:

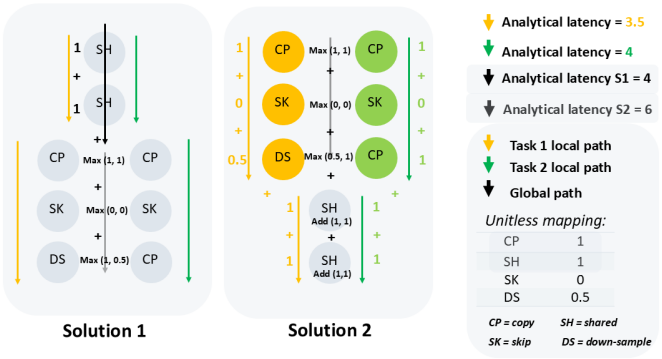


Fig. 3: Local and Global Analytical Latency Factor Mapping. Both solutions have the same local paths analytical latency. The global analytical latency prefers solution 1 since it can benefit from hardware parallelization without necessarily affecting parameters efficiency or individual tasks' policy decisions.

$$\mathcal{P}'(k) = \frac{\exp((G_k + \log(\pi_k))/\tau)}{\sum_{k\in\{0,1,2\}} \exp((G_k + \log(\pi_k))/\tau)} \tag{1}$$

where $k \in \{0, 1, 2\}$ represents the three operator options. The standard Gumbel distribution [21] $G_k \sim Gumbel(0, 1)$ is used to draw i.i.d (independent and identically distributed) samples where $\pi = [\pi_0, \pi_1, \pi_2]$ is the policy distribution ($\pi_0 + \pi_1 + \pi_2 = 1$), and τ is the temperature parameter. We propose a dynamic objective function that leverages both local and global ALF to guide policy-decisions training. The objective is to minimize:

$$\mathcal{L}_{total} = \sum \lambda_i \mathcal{L}_i + \lambda_{reg} \cdot \big(\alpha \cdot L_{Lat.} + (1 - \alpha) \cdot L_{reg}\big) \tag{2}$$

where λ_i are task weights, $\mathcal{L}_i$ are the individual task losses, and λ_{reg} is a scalar regularizer. To dynamically adjust the focus between latency optimization and parameters efficiency, we introduce an adaptive coefficient, α. If a user-defined local ALF requirement is met, we set α to zero, allowing $\mathcal{L}_{reg}$ to focus on parameters efficiency (through layers-sharing promotion). Otherwise, we set α to 1 which activates the latency-aware term, $L_{Lat.}$, penalizing policies exceeding analytical latency requirements. The latency-focused term is:

$$\mathcal{L}_{Lat.} = \sum_{i\leq N} \sum_{l\leq L} \left\{ \mathcal{GL} * (\exp^{\mathcal{P}_l^{t_i}(1)+\mathcal{P}_l^{t_i}(0)-\mathcal{P}_l^{t_i}(2)}) \right\} \tag{3}$$

$\mathcal{GL}$ is the global ALF. As gradient-descent step is applied to each task separately, the global factor can capture the latest changes from previous tasks' updates and inform the next task for a better global analytical latency. The term discourages copying and sharing the original operator. Instead,

it opts for aggressive down-sampling and unity. Meanwhile, the parameters efficient term [4] is:

$$\mathcal{L}_{reg} = \sum_{i \leq N} \sum_{l \leq L} \frac{L-l}{L} \Bigg\{ \ln\big(1 + \exp^{\mathcal{P}_l^{t_i}(1) - \mathcal{P}_l^{t_i}(0)}\big) + \ln\big(1 + \exp^{\mathcal{P}_l^{t_i}(2) - \mathcal{P}_l^{t_i}(0)}\big) \Bigg\} \tag{4}$$

The exponential nature of $\mathcal{L}_{Lat.}$ leads to aggressive updates during the initial training phase, driving the model towards low-analytical-latency configurations, but potentially sacrificing accuracy. Once the analytical latency requirement is met, the more linear updates from $\mathcal{L}_{reg}$ allow the model to gradually explore solutions and progressively increase capacity. As training progresses, $\mathcal{L}_{reg}$ may eventually push the model beyond the analytical latency limit, triggering a reactivation of $\mathcal{L}_{Lat.}$. This brings the model back into the low-latency design space region, but with a different configuration—one that has a higher starting capacity and better accuracy than before. This iterative process allows for deeper exploration of solutions near the aggressive low-analytical-latency boundary.

C. Capacity Increase and Conflict Handling

The Conflict Definition: Equation (2) can suffer from potential tasks interference. Let us assume $K \geq 2$ distinct tasks, each associated with a loss function $\mathcal{L}_i(\theta)$, where θ represents a shared set of parameters. The objective is to find an optimal θ that minimizes losses across all tasks. Let us also assume that $\sum_i \lambda_i = 1$. Taking the first term of (2), we have

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^m} \left\{ L_0(\theta) \triangleq \frac{1}{K} \sum_{i=1}^{K} L_i(\theta) \right\}. \tag{5}$$

Let $g_i = \nabla L_i(\theta)$ represent the gradient for task i. The averaged gradient across all tasks becomes $g_0 = \nabla L_0(\theta) = \frac{1}{K} \sum_{i=1}^{K} g_i$. Using a learning rate $\alpha \in \mathbb{R}^+$, the steepest descent update is $\theta \leftarrow \theta - \alpha g_0$. However, g_0 can conflict with individual gradients, meaning that for some task i, $\langle g_i, g_0 \rangle < 0$. When this conflict is significant, g_0 can degrade performance on task i. To handle interference, we combine a mitigation and removal strategy. The latter option is only possible after the second phase of training where the policies are finalized. Before starting the third phase, we calculate conflict scores for each convolutional layer shared between two or more tasks. Then, we allow up to 'n' layers to be transformed from shared to task-specific. By using task-specific layers, the gradients conflict is completely removed. The conflict scores are calculated by doing forward pass and back-propagation for $\mathcal{I}$ iterations. Mathematically, we utilize a layer-wise S-conflict score [22]. Let the shared model parameters θ_{sh} consist of n layers, i.e., $\theta_{\text{sh}} = \{\theta_{\text{sh}}^{(k)}\}_{k=1}^{n}$, where $\theta_{\text{sh}}^{(k)}$ represents the k^{th} shared layer. The gradient of task i with respect to the k^{th} shared layer, denoted by $\mathbf{g}_i^{(k)}$, is the vector of partial derivatives of $\mathcal{L}_i$ with respect to the parameters of $\theta_{\text{sh}}^{(k)}$. We Let $\phi_{ij}^{(k)}$ represent the angle between $\mathbf{g}_i^{(k)}$ and $\mathbf{g}_j^{(k)}$. The definition for layer conflict and S-conflict score as found in [22] is:

Definition 1 (Layer-wise Conflicting Gradients): The gradients $\mathbf{g}_i^{(k)}$ and $\mathbf{g}_j^{(k)}$ $(i \neq j)$ are considered conflicting if $\cos \phi_{ij}^{(k)} < 0$.

Definition 2 (S-Conflict Score): For any $-1 < S \leq 0$, the S-conflict score for the k^{th} shared layer is the number of task pairs (i, j) $(i \neq j)$ such that $\cos \phi_{ij}^{(k)} < S$, denoted as $s^{(k)}$. The top 'n' S-conflict score layers will be transformed to task-specific. The remaining shared layers will be handled through gradients modifications as in [23]. We find an update vector within a local region centered around the average gradient g_0 of the shared layers added to our overall latency-Aware loss, while minimizing conflict among tasks.

While we combine elements from both [23] and [22], our approach differs in: *1)* our loss is not an average gradient with equal weights but rather, a dynamic one that changes based on the analytical latency requirements. *2)* the gradients conflict handling is done on a layer level and is compatible with not only pure hard parameter-sharing backbones but also for backbones with various branching decisions where sharing occurs and disappear in different parts of the architecture. *3)* we enforce task-specific copies for BatchNorm layers. *4)* conflict handling in shared layers is allowed at any phase of the three training stages. *5)* in the third training stage, we train model parameters from scratch.

IV. Evaluation

For our experiments, we follow AutoMTL [4] for all evaluation metrics, task losses, hyperparameters, and data augmentation. On CityScapes [24] and NYUv2 [25] datasets, we optimize Deeplab models with ResNet-34 [26] and MobileNetV2 [27] backbones, and Atrous Spatial Pyramid Pooling (ASPP) task-specific heads. We additionally evaluate on the expanded form of the structurally reparametrizable MobileOne [28] backbone.

Backbone	Approach	Sem.Segm. ↑			Depth Est. ↑			All tasks ↑			Params ↓	
		ONNX	TRT	QNN	ONNX	TRT	QNN	ONNX	TRT	QNN	Rel.	Rel2.
ResNet-34 [26]	Baseline										-49.01	-48.19
	+LA-MTL-C	**15**	**23**	**18**	**15**	**11**	-49	**20**	**22**	**19**	**-68.88**	-68.14
	+LA-MTL-A	**40**	**45**	**27**	**28**	**34**	**20**	**33**	**39**	**33**	**-76.03**	-75.22
MobileNetV2 [27]	Baseline										-45.68	+4.43
	+LA-MTL-C	-11	**2**	1	-15	-17	**9**	-1	**2**	**3**	**-68.01**	+10.83
	+LA-MTL-A	-11	0	-19	0	**11**	1	**1**	**3**	**4**	**-87.91**	+14.22

TABLE II: **Latency Optimization on CityScapes**. Latency speedup and parameters achieved with LA-MTL (A: Aggressive, C: Conservative) compared to the AutoMTL baseline [4].

.

Backbone	Approach	Sem.Segm. ↑			Depth Est. ↑			Normal ↑			All tasks ↑			Params ↓	
		QNNX	TRT	QNN	QNNX	TRT	QNN	QNNX	TRT	QNN	QNNX	TRT	QNN	Rel.	Rel2.
ResNet-34 [26]	Baseline													-49.25	-48.47
	+LA-MTL-C	**19**	**21**	**20**	**26**	**27**	**32**	-2	**6**	0	**19**	**20**	**18**	**-62.02**	-61.06
	+LA-MTL-A	**35**	**39**	**32**	**44**	**62**	**61**	**18**	**40**	**34**	**44**	**48**	**44**	**-84.77**	-83.27
MobileNetV2 [27]	Baseline													-40.46	-10.52
	+LA-MTL-C	-3	-3	**1**	**1**	**20**	**6**	**15**	-18	**1**	**2**	**1**	**4**	**-75.75**	+6.44
	+LA-MTL-A	-23	**1**	0	**5**	-14	-2	-16	-28	**3**	**7**	-4	**6**	**-82.76**	+12.05

TABLE III: **Latency Optimization on NYUv2**. Latency speedup and parameters achieved with LA-MTL (A: Aggressive, C: Conservative) compared to the AutoMTL baseline [4].

For the parameters reduction metric, we calculate it relative to a *single task approach* in two ways. 'Rel.' follows the SOTA [4] by convention, and considers share-choice and copy-choice only and 'Rel2.' considers share-choice, copy-choice,

and skip-choice. We utilize two variants of our latency-aware loss. The conservative variant sets a latency target of 0.5 times the analytical latency of the original backbone for each task. In contrast, the aggressive variant imposes a stricter target, aiming for 0.2 times the original backbone's analytical latency. These latency constraints are applied uniformly across all tasks and layers, along with a consistent analytical mapping for all layers and tasks. However, we permit individual layer mappings to deviate from this uniform mapping if the direct neighborhood of a layer has significantly higher FLOPs, differing by orders of magnitude.

A. Latency Optimization

We measure the latency (in milliseconds) of a single experiment by averaging 20 consecutive inferences of the MTL backbones (excluding task heads). Deployments are executed on two distinct platforms: Nvidia Jetson AGX Orin [29] using ONNX/TensorRT runtime, and Qualcomm Snapdragon Gen 2 mobile development kit [30] utilizing Qualcomm AI Direct SDK QNN runtime [31]. Tables II and III present the reduction metrics for latency and parameters on ResNet-34 and MobileNetV2 backbones, on CityScapes and NYUv2 datasets, respectively. The observations from Table II on CityScapes and Table III on NYUv2 provide comprehensive insights into the performance of LA-MTL variants using ResNet-34 and MobileNetV2 backbones. For CityScapes, the conservative approach (LA-MTL-C) with ResNet-34 achieves consistent latency reductions across most tasks and runtimes, with parameter reductions up to 68.88%, though there is a notable 49% latency increase for the QNN runtime in depth estimation. The aggressive approach (LA-MTL-A) demonstrates even greater latency reductions, especially for semantic segmentation and depth estimation, achieving up to 45% faster latency and parameter reductions of 76.03%. For MobileNetV2, the conservative approach shows mixed latency results but achieves a parameter reduction of 68.01%, while the aggressive approach indicates higher latencies for individual tasks but faster overall latency when all tasks are activated simultaneously, achieving the highest parameter reduction of 87.91%. For the NYUv2 dataset, the conservative approach (LA-MTL-C) with ResNet-34 shows latency reductions across semantic segmentation, depth estimation, and normal tasks, with parameter reductions of 62.02%. The aggressive approach (LA-MTL-A) achieves even more significant latency reductions, with up to 62% reduction for depth estimation and 40% for normal tasks, and parameter reductions of 84.77%. For MobileNetV2, the conservative approach again shows mixed results, with some latency increases, but achieves a parameter reduction of 75.75%. The aggressive approach shows higher latencies for individual tasks but achieves up to 82.76% parameter reduction. These results showcase the effectiveness of LA-MTL approaches in optimizing latency and memory footprint across different datasets, backbones and runtimes. The conservative approach generally provides balanced improvements, while the aggressive approach can achieve more substantial reductions but may involve trade-offs in individual task latencies. These outcomes can be attributed to several factors, including the blackbox compiler optimizations, which can vary in their effectiveness depending on the obtained MTL architecture. Additionally, utilization of hardware resources is crucial, as different optimization strategies may lead to varying degrees of hardware utilization efficiency. This highlights the importance of choosing the appropriate optimization strategy based on the target platform, runtime and performance goals.

B. Task Metrics and SOTA Comparison

Optimizing for latency often directly impacts task metrics, potentially degrading them. Therefore, our focus extends to further boosting task metrics performance achieved with LA-MTL. For ResNet-34, the non-latency-aware AutoMTL [4] loss clearly benefited from interference handling and increased capacity (Table VI) across all task metrics. The results from the table indicate that the gradient-modified AutoMTL model achieves superior performance in semantic segmentation compared to other state-of-the-art (SOTA) models. Specifically, it achieves a mean Intersection over Union (mIoU) of 44.63% and pixel accuracy of 75.07%. This performance surpasses other well-known models such as**Cross-Stitch** [2], **Sluice** [32], **NDDR-CNN** [33], **MTAN** [1], **DEN**, [34], and **AdaShare** [3], which report lower values for mIoU and pixel accuracy. The gradient-modified AutoMTL model not only excels in accuracy but also manages to maintain a relatively low parameter count, making it an efficient choice for multi-task learning applications. Additionally, the LA-MTL-C model demonstrates remarkable parameter efficiency, achieving a parameter reduction of -68.88% (Rel.) and -68.14% (Rel2.), while still maintaining competitive performance in both semantic segmentation and depth estimation tasks. This level of parameter reduction is the highest among the models compared, indicating that LA-MTL-C is highly efficient in terms of resource usage. For MobileNetV2, the gradient-modified LA-MTL-C model particularly stands out with the best depth estimation accuracy, achieving a 1.25 accuracy of 46.51. This performance is achieved while also maintaining significant parameter reductions, showcasing the model's ability to balance between performance and efficiency. In contrast, other models like AdaShare and the original AutoMTL, while showing strong performance, do not achieve the same level of parameter efficiency. Overall, these comparisons, also proved in VII highlight the effectiveness of the proposed approach in achieving a balance between high task performance and parameter efficiency. The gradient-modified AutoMTL and LA-MTL-C models, in particular, outperform other SOTA methods in various metrics and tasks, making them highly suitable for practical applications where resource constraints are a concern.

C. Ablation Study

Table V shows that our global factor is able to promote further parameters efficiency, faster local paths latency, and competitive metrics. As for tasks metrics, they were competitive to baseline (drop $< 1\%$) and surpassed it in $\delta 1.25^2$

CityScapes	Params Reduction		Sem. Segm.					Depth Estimation					
	Rel. ↑	Rel2 ↑	mIoU ↑	P.Acc ↑	Lat1^a ↑	Lat2^a ↑	Lat3^a ↑	1.25 ↑	1.25^2 ↑	1.25^3 ↑	Lat1^a ↑	Lat2^a ↑	Lat3^a ↑
MobileOne													
AutoMTL [4]	35.07	**21.57**	**25.20**	**63.62**	0.0	0.0	0.0	**46.18**	**74.51**	**87.15**	0.0	0.0	0.0
LA-MTL-C	49.65	29.22	24.60	63.45	-7	-7	-1	45.31	73.92	86.93	10	-7	-2
LA-MTL-A	**66.05**	**38.32**	24.36	63.38	9	16	4	33.87	57.33	74.40	16	-49	21

TABLE IV: MobileOne expanded form with no capacity/gradients modifications on CityScapes dataset. Values are an average of 4 random seeds. All metrics in %; Latencies in ms; ↑ = higher is better. P.Acc = Pixel Acc.; Lat1, Lat2, Lat3 = ONNX, TRT, QNN latency reduction.

Method	Params Reduction ↑		Semantic Seg.			Depth Estimation					
						Error ↓		δ, within ↑			
	Rel. (%)	Rel2. (%)	mIoU ↑	Pixel Acc. ↑	Lat1. ↓	Abs.	Rel.	1.25	1.25^2	1.25^3	Lat1. ↓
Conserv. w/ GAL	33.30	33.21	33.47	71.63	25.46	0.0232	0.399	64.42	83.11	91.40	24.43
Conserv. w/o GAL	23.86	23.84	34.06	**71.81**	26.49	**0.0223**	0.4343	64.24	82.29	90.47	26.49
Aggres. w/ GAL	**52.47**	**52.05**	33.03	71.53	**21.60**	0.0219	0.4116	**64.47**	82.73	90.95	**21.60**
Aggres. w/o GAL	31.67	31.56	**34.16**	71.67	25.04	0.0237	**0.3864**	64.30	**83.64**	**91.62**	25.06
AutoMTL [4]	26.54	26.23	33.71	71.76	28.07	0.0229	0.3957	64.61	83.48	91.51	28.11
Conserv. w/ GAL	**37.86**	**36.66**	**32.64**	**71.55**	22.04	**0.0222**	0.4171	**63.52**	**82.32**	**90.61**	**20.74**
Conserv. w/o GAL	26.24	25.36	31.58	71.42	22.13	0.0225	0.4385	62.75	81.47	90.05	22.12
Aggres. w/ GAL	44.96	43.69	31.74	71.42	**18.71**	0.0227	0.4287	62.53	81.61	90.21	21.21
Aggres. w/o GAL	34.56	33.68	32.51	71.58	22.01	0.0228	0.4519	62.97	81.07	89.50	22.35
AutoMTL [4]	36.11	35.71	33.69	71.65	25.64	0.0229	**0.4113**	64.40	83.08	91.01	25.76
Conserv. w/ GAL	**36.13**	**35.42**	**32.94**	**71.84**	29.56	0.0225	0.4361	63.21	82.12	90.46	29.63
Conserv. w/o GAL	31.28	30.57	32.59	71.76	29.81	0.0231	0.3981	63.77	**83.12**	**91.30**	**27.10**
Aggres. w/ GAL	39.99	38.97	31.38	71.62	**27.06**	0.0245	0.4145	62.20	82.26	90.84	27.34
Aggres. w/o GAL	36.39	35.68	31.55	71.52	29.27	**0.0222**	0.4159	63.82	82.49	90.67	29.93
AutoMTL [4]	44.50	44.11	33.35	71.71	30.43	**0.0216**	**0.4476**	**64.20**	81.90	89.92	31.26

TABLE V: Global ALF Effect (CityScapes, Resnet34) and ONNX latencies in ms.

and $\delta 1.25^3$. Our ALF method is able to deliver- again-latency improvement in average for the expanded MobileOne backbone (Table IV). While mostly losing less than 1 p.p. on tasks metrics, we achieved up to 21% faster latency and simultaneously removed 38.32% parameters compared to just 21.57% baseline.

CityScapes	# Params ↓		Sem. Segm.		Depth Estimation		
	Rel. (%)	Rel2. (%)	mIoU ↑	Pixel Acc. ↑	1.25 ↑	1.25^2 ↑	1.25^3 ↑
ResNet-34							
Single-Task	-	-	36.5	73.8	57.5	76.9	87.0
Multi-Task	-50.0	-	42.7	68.1	58.8	80.5	89.9
Cross-Stitch [2]	+0.0	-	40.3	74.3	70.0	86.3	93.1
Sluice [32]	+0.0	-	39.8	74.2	68.9	85.8	92.8
NDDR-CNN [33]	+3.5	-	41.5	74.2	69.9	86.3	93.0
MTAN [1]	+20.5	-	40.8	74.3	71.0	86.3	92.8
DEN [34]	-44.0	-	38.0	74.2	68.2	84.5	91.6
AdaShare [3]	-50.0	-	40.6	74.7	**71.4**	**86.8**	93.1
AutoMTL [4]	-32.3	-	42.8	74.8	70.0	86.6	**93.4**
+ Grad. Mod.	-41.38	-40.28	**44.63**	**75.07**	70.11	86.16	92.98
LA-MTL-C	-68.88	-68.14	39.31	74.27	66.42	84.23	91.98
+ Grad. Mod.	-58.17	-57.41	42.07	74.59	69.19	84.78	92.04
LA-MTL-A	**-76.03**	**-75.22**	39.21	74.22	63.67	83.31	91.65
MobileNetV2							
Single-Task	-	-	25.9	63.5	32.1	71.1	86.5
Multi-Task	-50.0	-	26.3	63.4	33.6	66.5	82.9
AdaShare [3]	-50.0	-	**26.7**	61.2	42.1	71.6	84.3
AutoMTL [4]	-33.5	-	25.8	**63.7**	44.4	**74.8**	**87.3**
LA-MTL-C	**-66.55**	+14.51	22.81	63.2	39.57	65.98	80.32
+ Grad. Mod.	-50.95	+7.25	24.01	63.03	**46.51**	72.02	85.3
LA-MTL-A	**-90.50**	+14.16	25.99	62.55	39.95	62.91	77.06

TABLE VI: SOTA Comparison on the CityScapes dataset with LA-MTL.

V. Conclusion

This paper proposed a latency-aware MTL optimization method that attempts maintaining good tasks performance, parameters efficiency, and lower latency for edge inference. For any given convolutional architecture, our modular framework

CityScapes	# Params ↓		Sem. Segm.		Depth Estimation			Normal		
	Rel.(%)	Rel2.(%)	mIoU ↑	Pixel Acc. ↑	δ, within ↑			θ, within ↑		
					1.25	1.25^2	1.25^3	θ11°	θ22°	θ30°
ResNet-34										
Single-Task	-	-	26.5	58.2	57.8	85.8	96.0	29.4	**72.3**	**87.3**
Multi-Task	- 67.68	-	22.2	54.4	60.9	87.7	**96.7**	32.2	70.5	84.8
Cross-Stitch	+0.0	-	25.4	57.6	61.4	88.4	95.5	41.4	67.7	80.4
Sluice	+0.0	-	23.8	56.9	61.9	88.1	96.3	38.9	69.0	81.4
NDDR-CNN	+1.81	-	21.6	53.9	55.7	83.7	94.8	37.4	70.9	83.1
MTAN	+0.55	-	26.0	57.2	**62.7**	87.7	95.9	**43.7**	70.5	81.9
DEN	-63.80	-	23.9	54.9	22.8	62.4	88.2	36.0	70.6	83.4
AdaShare	-67.68	-	24.4	57.8	61.3	**88.5**	96.5	42.3	68.9	80.5
AutoMTL	-50.64	-49.94	26.33	57.97	61.29	87.77	96.38	37.76	70.68	83.57
LA-MTL-C	-71.85	-70.88	**26.92**	**58.58**	50.96	80.99	93.74	38.52	**72.58**	85.11
LA-MTL-A	**-87.91**	**-86.41**	23.79	55.53	49.31	79.68	93.02	32.79	72.17	86.40
MobileNetV2										
AutoMTL [4]	-40.46	**-10.52**	**18.69**	**47.52**	**53.85**	**82.76**	**94.22**	**27.10**	72.04	86.65
LA-MTL-C	-75.75	+6.44	17.70	46.67	51.08	81.12	93.49	26.44	72.10	86.77
LA-MTL-A	**-82.76**	+12.05	17.17	46.40	47.53	77.07	91.17	20.50	**72.27**	**87.03**

TABLE VII: Performance comparison across various multi-task learning methods.

is designed to work independently with little manual effort. The modest capacity increase along with gradients conflict mitigation boosted SOTA AutoMTL to have improved task metrics outperforming other related works. The latency aware optimization along with the analytical latency factor delivered a superior parameters efficiency surpassing SOTA. Tested on Cityscapes and NYUv2 datasets and on Resnet, MobileNet, and MobileOne architectures, LA-MTL is indeed able to navigate through the design space while finding solutions with improved latency and competitive task metrics.

References

[1] S. Liu, E. Johns, and A. J. Davison, "End-to-end multi-task learning with attention," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1871–1880.

[2] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," 2016.

[3] X. Sun, R. Panda, R. Feris, and K. Saenko, "Adashare: Learning what to share for efficient deep multi-task learning," 2020.

[4] L. Zhang, X. Liu, and H. Guan, "Automtl: A programming framework for automating efficient multi-task learning," 2022.

[5] A. M. Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "Running hardware-aware neural architecture search on embedded devices under 512mb of ram," *IEEE International Conference on Consumer Electronics*, 2024.

[6] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast hardware-aware neural architecture search," 2020.

[7] H. Bouzidi, H. Ouarnoughi, S. Niar, and A. A. E. Cadi, "Performance prediction for convolutional neural networks in edge devices," 2020.

[8] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," 2019.

[9] N. Sinha, A. E. R. Shabayek, A. Kacem, P. Rostami, C. Shneider, and D. Aouada, "Hardware aware evolutionary neural architecture search using representation similarity metric," *arXiv.org*, 2023.

[10] Z. Li, M. Paolieri, and L. Golubchik, "Inference latency prediction at the edge," 2022.

[11] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," 2019.

[12] S. Zeng, H. Sun, Y. Xing, X. Ning, Y. Shan, X. Chen, Y. Wang, and H. Yang, "Black box search space profiling for accelerator-aware neural architecture search," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 518–523.

[13] H. Lee, S. Lee, S. Chong, and S. J. Hwang, "Help: Hardware-adaptive efficient latency prediction for nas via meta-learning," 2021.

[14] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020. [Online]. Available: https://arxiv.org/abs/1905.11946

[15] H. Liang, Z. Fan, R. Sarkar, Z. Jiang, T. Chen, K. Zou, Y. Cheng, C. Hao, and Z. Wang, "M^3vit: Mixture-of-experts vision transformer for efficient multi-task learning with model-accelerator co-design," 2022. [Online]. Available: https://arxiv.org/abs/2210.14793

[16] X. Zhou, Y. Gao, C. Li, and Z. Huang, "A multiple gradient descent design for multi-task learning on edge computing: Multi-objective machine learning approach," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 121–133, 2022.

[17] A. Javaloy and I. Valera, "Rotograd: Gradient homogenization in multitask learning," 2022. [Online]. Available: https://arxiv.org/abs/2103.02631

[18] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," 2020. [Online]. Available: https://arxiv.org/abs/2001.06782

[19] A. Gesmundo and J. Dean, "munet: Evolving pretrained deep neural networks into scalable auto-tuning multitask systems," 2022.

[20] T. Vu, Y. Zhou, C. Wen, Y. Li, and J. Frahm, "Toward edge-efficient dense predictions with synergistic multi-task neural architecture search," *IEEE Workshop/Winter Conference on Applications of Computer Vision*, 2023.

[21] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," 2017. [Online]. Available: https://arxiv.org/abs/1611.01144

[22] G. Shi, Q. Li, W. Zhang, J. Chen, and X.-M. Wu, "Recon: Reducing conflicting gradients from the root for multi-task learning," 2023. [Online]. Available: https://arxiv.org/abs/2302.11289

[23] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu, "Conflict-averse gradient descent for multi-task learning," 2024. [Online]. Available: https://arxiv.org/abs/2110.14048

[24] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3213–3223, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:502946

[25] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *Proceedings of the 12th European Conference on Computer Vision - Volume Part V*, ser. ECCV'12. Berlin, Heidelberg: Springer-Verlag, 2012, p. 746–760. [Online]. Available: https://doi.org/10.1007/978-3-642-33715-4_54

[26] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," 2017. [Online]. Available: https://arxiv.org/abs/1606.00915

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[28] P. K. A. Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan, "Mobileone: An improved one millisecond mobile backbone," 2023. [Online]. Available: https://arxiv.org/abs/2206.04040

[29] NVIDIA, "Nvidia jetson orin," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/?srsltid=AfmBOoo4VBZk9BMaYyKmT7nvGsMOhavegQB0W-db77zi7l3qj5WZVrXz, 2022, accessed: 2023-04-12.

[30] Qualcomm, "Snapdragon 8 gen 2 mobile platform," https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-2-mobile-platform, 2022, accessed: 2023-04-12.

[31] Q. Technologies, "Snapdragon® 8 gen 2 mobile hardware development kit," https://developer.qualcomm.com/hardware/snapdragon-8-gen-2-hdk, 2024.

[32] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, "Latent multi-task architecture learning," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33014822

[33] Y. Gao, J. Ma, M. Zhao, W. Liu, and A. L. Yuille, "Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 3200–3209.

[34] C. Ahn, E. Kim, and S. Oh, "Deep elastic networks with model selection for multi-task learning," 2019. [Online]. Available: https://arxiv.org/abs/1909.04860

SSpMV: A Sparsity-aware SpMV Framework Empowered by Multimodal Machine Learning

Shengle Lin[a], Chubo Liu[a*], Yan Ding[a], Joey Tianyi Zhou[b,c], Kenli Li[a] and Wangdong Yang[a*]
[a] College of Computer Science and Electronic Engineering, Hunan University, China
[b] Centre for Frontier AI Research, Agency for Science, Technology and Research, Singapore
[c] Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore
{lsl036, liuchubo, ding, lkl, yangwangdong}@hnu.edu.cn, zhouty@cfar.a-star.edu.sg

***Abstract*—Sparse Matrix-Vector Multiplication (SpMV) is an essential sparse operation in scientific computing and artificial intelligence. Efficiently adapting SpMV algorithms to diverse matrices and architectures requires a framework capable of accurately recognizing sparse patterns and selecting the optimal implementation. In this work, we introduce Sparsity-aware SpMV (SSpMV), a framework that integrates expert-designed features with multimodal representations to adaptively predict the best-performing algorithm and parameters. For this purpose, we design a multimodal neural network called MM-Adapter, to capture diverse modalities to represent the computational features of SpMV. Experimental results demonstrate that MM-Adapter achieves the highest accuracy of 81.05%, outperforming existing SpMV prediction models. Furthermore, SSpMV consistently delivers substantial performance improvements over state-of-the-art sparse libraries across various multi-core platforms.**

***Index Terms*—Sparse Matrix, SpMV, Machine Learning.**

TABLE I
FORMATS WITH HYPERPARAMETERS USED IN SSPMV

Format	Method	Params	Description
COO	OpenMP, *Lb*	SCH	SCH is the OpenMP scheduling mode, including Static (St), StaticChunk (StC), Dynamic (Dyn) and Guided (Gd). *Lb* denotes parallelism with a load-balanced partition by nnz.
CSR	OpenMP, *Lb*	SCH	
DIA	OpenMP, *Lb*	SCH	
ELL	OpenMP, *Lb*	SCH	
S-ELL	S-ELL (*Lb*)	SCH	
	SELL-c-σ	SCH, c, σ	c is the row chunk size that affects vector paddings. σ defines the reordering blocks of rows to reduce fillings while trying to keep locality.
	SELL-c-R	SCH, c	
BCSR	OpenMP, *Lb*	SCH, bR, bC	bR, bC control the block size by SIMD width and alignment.
CSR5	SIMD Intrinsics	σ_5, ω	ω is the size of SIMD unit. σ_5 is defined by cache capacity.

I. INTRODUCTION

With advancements in hardware and the exponential growth of data, sparse linear algebra has emerged as a cornerstone in circuit simulation, electronic design automation, and artificial intelligence [1]–[3]. A fundamental operation within sparse computing paradigms is Sparse Matrix-Vector Multiplication (SpMV), defined as $y = Ax$, which plays a pivotal role in solving sparse linear systems [4]. The SpMV kernel is a critical component of iterative solvers such as Generalized Minimal Residual (GMRES) and Preconditioned Conjugate Gradient (PCG), which involve repeated operations on matrices with fixed sparse structures [5], [6]. Therefore, optimizing the performance of SpMV across diverse platforms is essential due to its significant computational demands.

Achieving consistently high-performance SpMV kernels on multi-core platforms remains challenging due to irregular memory access patterns and limited utilization of vector instructions. As listed in Table I, previous approaches often introduce specialized formats, such as SELL-c-σ [7], CSR5 [8], BCSR [9], tailored to specific hardware architectures to enhance computational efficiency. However, as illustrated in Fig. 1, no single format optimally suits all matrices, as performance depends on specific sparse patterns. Some recent studies prefer to leverage machine learning models, such as decision trees (DT) [10], [11], support vector machines (SVM) [12], multilayer perceptrons (MLP) [13], and convolutional neural networks (CNN) [14], to adaptively select the best format for specific matrices. Alphasparse [15] automatically creates machine-designed GPU kernels based on the input pattern. While such frameworks consider many expert-designed matrix features, **the intricate interplay between sparse patterns and architectural characteristics, such as memory locality and vectorization, remains underexplored**. Furthermore, many studies rely on limited or synthetic datasets, which diminishes their applicability to real-world sparse systems, such as those found in circuit simulation problems.

To this end, we propose a Sparsity-aware SpMV framework (SSpMV) to address these issues. Specifically, SSpMV consists of two core components: 1) a suite of SpMV algorithms deeply optimized across different multi-core CPUs; and 2) an intelligent adapter, MM-Adapter, which employs a comprehensive set of multimodal representations to map a sparse pattern to its most efficient SpMV algorithm. Moreover, we utilize finite element software FreeFEM [16] and computational fluid dynamics software OpenFoam [17] to generate an extensive dataset of matrices, including a wide range of real-world scenarios. Our experimental results on over 6,000 sparse matrices reveal that SSpMV achieves an average speedup of $3.08\times$, with a maximum of $71.7\times$, compared to the Intel MKL [18]. The main contributions of this work are as follows:

* Corresponding authors.

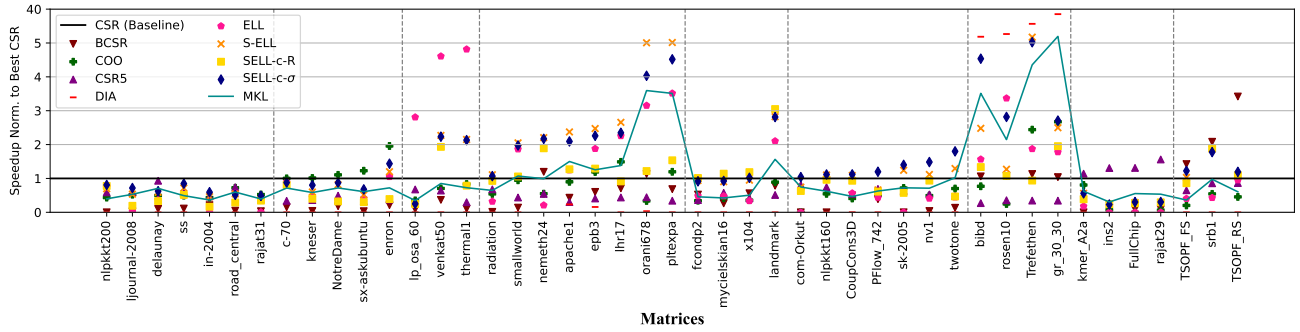


Fig. 1. Speedup of double-precision SpMV in different sparse formats and Intel MKL SpMV kernel compared to the best implementation of CSR format.

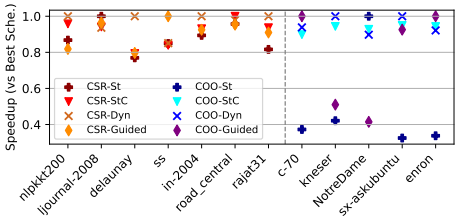
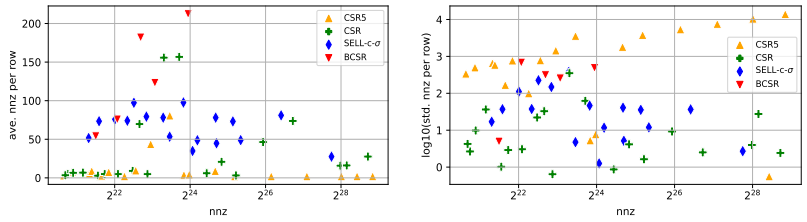


Fig. 2. Performance degradation of CSR and COO by different scheduling methods.



(a) Average number of *nnz* (b) Standard deviation of *nnz*

Fig. 3. Distribution of fastest method when concerning features of row nnz.

(1) We develop a high-performance Sparsity-aware SpMV framework that automatically maps a sparse matrix to its optimal SpMV implementation.
(2) We design a multimodal convolutional neural network, MM-Adapter, capable of extracting diverse modalities of a sparse pattern to represent the architectural features relevant to SpMV computation.
(3) We conduct extensive experiments on a large dataset of real-world sparse systems, demonstrating significant performance improvements achieved by SSpMV.

II. Background and Motivation

The SpMV operation ($y = Ax$) multiplies a sparse matrix A by a dense vector x, resulting in a dense output vector y. A wide variety of sparse storage formats have been developed to balance compression efficiency and SpMV performance. To comprehensively evaluate these algorithms and uncover the potential of various formats and parameters, we conduct extensive benchmarks on matrices from SuiteSparse Collection [19]. All methods listed in Table I are evaluated, where *COO*, *CSR*, *DIA* and *ELL* represent the four fundamental storage formats; *BCSR* [9], *S-ELL Series* [7] and *CSR5* [8] are vector-friendly optimized formats. The performance distributions reveal the following key insights:

1. The four basic formats still have their own performance advantages: As illustrated in Fig.1, each SpMV format excels with specific matrices, even for the four basic formats. Specifically, COO and CSR are well-suited for the sparse structures with irregular or fragmented nonzeros. In contrast, ELL outperforms S-ELL Series when row lengths are uniform, because balanced nonzero distributions allow ELL to achieve good memory efficiency without additional segmentation or reordering. DIA exhibits significant speedup for main-diagonalized nonzero patterns, with accelerations ranging from $11.4\times$ to $34.8\times$ in examples. In addition, the performance of *Intel MKL dcsr_SpMV* is recorded, revealing that it is suboptimal for the majority of matrices.

2. Picking the suitable parameters is important for SpMV performance: Fig.2 shows the significant impact of different OpenMP scheduling policies (*St, StC, Dyn* and *Guided*) on SpMV performance. The performance degradation, measured relative to the optimal scheduling strategy, reveals that CSR fluctuates between $0.77\times$ and $1\times$, while COO can suffer up to a $3.1\times$ slowdown. This disparity arises because the COO parallelizes based on individual nonzeros, making it highly sensitive to scheduling policies. Given the variability of sparse patterns, identifying a universally optimal scheduling strategy is inherently challenging, let alone tuning additional configuration parameters.

3. Existing expert-designed features are insufficient to characterize sparse patterns: We consider two widely used features about the number of non-zeros (*nnz*) per row: the average and the standard deviation [10], [11], [20]. From Fig.3a, we observe that BCSR and SELL-c-σ perform better on datasets with higher row *nnz*, while CSR and CSR5 generally favor more sparser patterns. Fig.3b reveals a cleaner distribution pattern for the CSR5 format, showing its marked advantage in cases with extreme imbalance. However, when the workload decreases and *nnz* becomes similar between rows, it is difficult to predict which method will be the fastest.

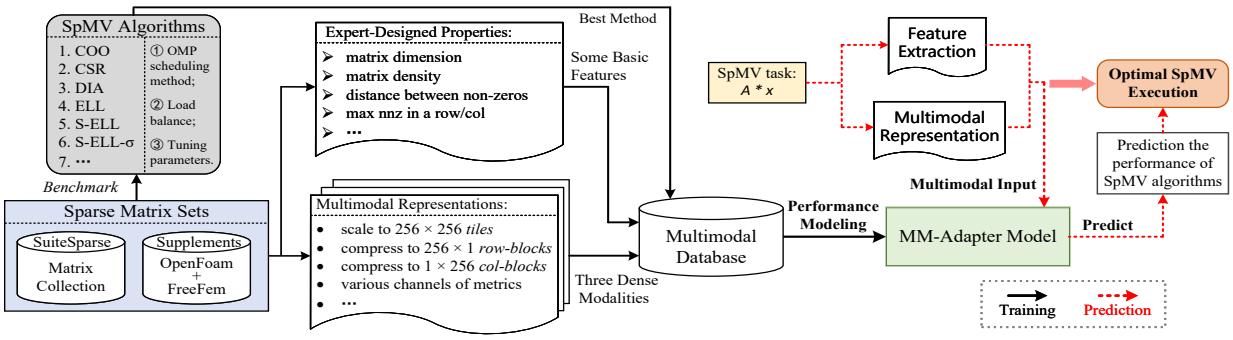


Fig. 4. Overview of the Sparsity-aware SpMV framework.

Granularity	Features	Description
Matrix	row, col, nnz, density	Depict the overall Matrix
	nnz_ratio_on_main_diag	Capture the distribution of nnz on diagonals
	ave_nnz_distance	Capture the locality to the input vector
Rows	nz_R, max_R, μ_R, σ_R, G_R, P_R	Determine the row scheduling method and paddings
Columns	nz_C, max_C, μ_C, σ_C, G_C, P_C	Determine the memory access pattern to input vector
Tiles	nz_T, max_T, μ_T, σ_T, G_T, P_T	Describe the balance of nnz across Tiles
	uniqR, uniqC	Describe the locality inside a Tile
	reuseR, reuseC	Describe the data reuses probability across Tiles
Row Block	nz_{RB}, max_{RB}, μ_{RB}, σ_{RB}, G_{RB}, P_{RB}	Determine cache utilization and vectorization
Col Block	nz_{CB}, max_{CB}, μ_{CB}, σ_{CB}, G_{CB}, P_{CB}	Determine cache utilization and cache-line alignment

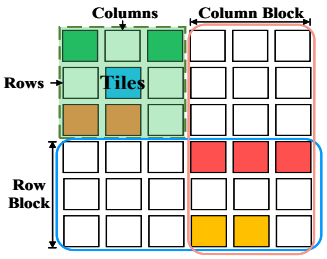


Fig. 5. Expert-designed features and their descriptions. The right figure shows the different granularities for feature extraction.

Overall, a comprehensive exploration of all formats and parameters enables the SpMV kernel to exploit the performance potential, but the intricate relationship between sparse patterns and architectural features makes it challenging to tell optimal implementation based solely on expert-designed features.

III. Sparsity-aware SpMV Framework

A. Overview of SSpMV

Given the above observation and analysis, we propose SSpMV, a Sparsity-aware SpMV framework that selects the most efficient format and implementation for a given sparse matrix. Fig.4 provides an overview of SSpMV. First, the framework generates kernels for all formats listed in Table I based on hardware characteristics such as the length of SIMD units and cache line size. For each input matrix, SSpMV extracts expert-designed features and multimodal representations to capture the architectural features relevant to SpMV computation. Through benchmarking on extensive datasets, it constructs a feature-label pair for each matrix. To enhance sparsity awareness, we develop a multimodal machine learning model, named *MM-Adapter*, which leverages the multimodal features of matrix sparsity to predict the most suitable SpMV implementation. As a result, SSpMV ensures optimal selection among various formats and parameters, maximizing SpMV performance for different sparse matrices across platforms.

B. Expert-designed Features

As a sparsity-aware framework, SSpMV begins with a feature extraction process to collect a broad range of expert-designed features. Some features directly characterize the overall matrix, such as matrix size, density, the proportion of *nnz* on the main diagonals, and the average distance. Other carefully selected features can reflect memory access patterns, vector paddings, and data skew, forming the basis for accurate performance modeling of SpMV operation.

Fig.5 summarizes all the expert-designed features used in SSpMV. Specifically, we partition the sparse matrix at different granularities to compute the six fine-grained features from each perspective. In addition to rows and columns, we partition the sparse matrix into 256×256 tiles, 256×1 row blocks, and 1×256 column blocks, respectively. Each partition is referred to as a "bucket", and the following six fine-grained expert-designed features are calculated:

1) *nz*: Ratio of buckets that are not zero;
2) *max*: Maximum of *nnz* between buckets;
3) μ: Mean of *nnz* between buckets;
4) σ: Standard deviation of *nnz* between buckets;
5) G: The *Gini coefficient* of the nnz among buckets ($0 \sim 1$);
6) P: The *p-ratio* of the *nnz* among buckets. P-ratio means that $p\%$ of the buckets have $(1-p)\%$ nonzeros ($0 \sim 0.5$).

The first four features are commonly used to characterize the statistical properties of data distributions [9], [11], [21]. The last two features measure equilibrium: in a perfectly balanced scenario, the p-ratio is 0.5 and the Gini coefficient is 0. As described in Fig.5, these 40 features provide a preliminary assessment of the sparse structures.

C. Multimodal Representations

Representing the matrix sparsity typically encounters two significant challenges. First, sparse matrices vary greatly in scale, making it difficult to effectively apply fixed-resolution convolutional neural networks (CNNs). Second, the complex

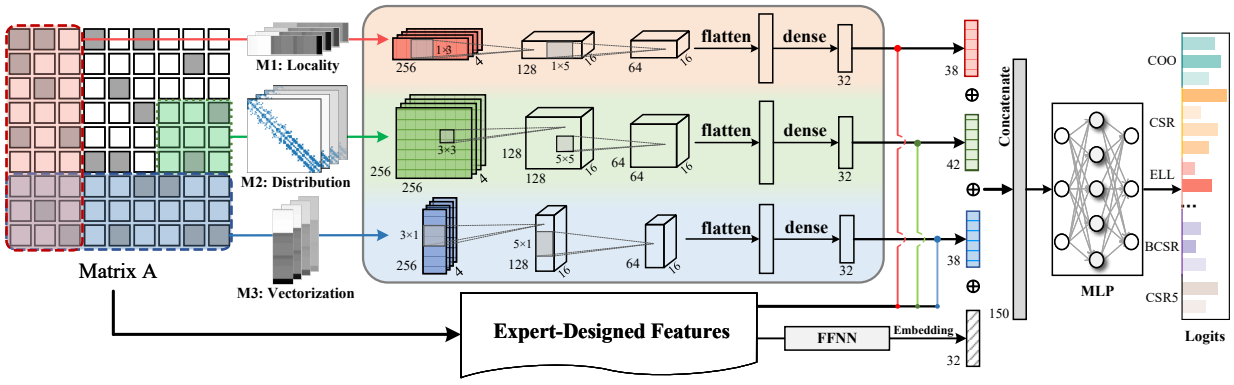


Fig. 6. Details of MM-Adapter. Multimodal representations are extracted by CNNs and fused with expert-designed features to model SpMV performance.

sparsity of matrices results in unique preferences for memory access patterns, scheduling strategies, and other architectural factors, which cannot be fully captured by existing approaches [14], [21]. Inspired by multimodal machine learning, we introduce novel multimodal representations that comprehensively encapsulate the diverse properties of sparse structures.

The unique challenges in SpMV have led us to propose a new interpretation of multimodality in the context of SpMV operation. Unlike traditional multimodal learning that deals with different data types, our approach focuses on characterizing the diverse architectural features relevant to sparse matrix computation. Specifically, we introduce three distinct modalities that reflect hardware preferences: **Locality**, **Distribution**, and **Vectorization**, each addressing specific requirements of SpMV kernels. We obtain these three types of multimodal representations by compressing the sparse matrices with granularity aligned to the scale of expert feature extraction. As shown in Fig.6, the **Locality** modality (M_1, red block) reflects the access pattern to the input vector, reflecting memory locality during computation. The **Distribution** modality (M_2, green tile) captures a snapshot of the workload within the matrix, providing insights for load balancing strategies. Finally, the **Vectorization** modality (M_3, blue block) represents latent features of a sparse matrix when row vectorization is applied.

Indeed, the compression of each modal representation may result in the loss of structural details. To mitigate this issue, we design different channels for each modality to encode statistical information related to original sparse patterns:

- C_1: *Nnz* of each modality;
- C_2: Maximum *nnz* per row of each modality;
- C_3: Average *nnz* per row of each modality;
- C_4: Standard deviation of *nnz* per row of each modality.

The multi-channel multimodal representations enable SSpMV to comprehensively perceive matrix sparsity and effectively explore architectural properties critical to SpMV computation.

D. MM-Adapter Design

After extracting the three architecture-related modalities, we employ a multimodal machine learning model for feature fusion. Given the impressive success of CNNs in the image domain and the ability of multimodal representations to overcome matrix size variability, we select CNNs as the backbone of our *Multimodal Adapter* (MM-Adapter).

As illustrated in Fig.6, MM-Adapter requires four distinct inputs derived from the sparse matrix A: three architectural modality representations, and one expert-designed feature set. For the modalities associated with locality and vectorization, CNNs employ a series of *conv1D* layers to capture global structures, while for the distribution modality, *conv2D* layers are used to extract local patterns. Subsequently, the expert-designed features, which serve as **the fourth artificial modality**, are integrated as an augmentation to the latent features, aligned with the specific partitioning granularity of each modality. A feed-forward neural network (FFNN) then processes all artificial features and transforms them into a dense embedding. Finally, a multilayer perceptron (MLP) takes the concatenated embedding as a fused input and maps the given matrix to the optimal SpMV implementation.

To construct the matrix labels, we compute the selection probability for each SpMV algorithm as follows:

$$P_i = \frac{G_i}{G_1 + G_2 + ... + G_N}, \tag{1}$$

where G_i represents the GFlops achieved by the i-th algorithm. The optimal SpMV kernel is identified as the algorithm with the highest probability, as shown in the "Logits" of Fig.6. With this training methodology, the MM-Adapter can be easily adapted to any platform, and new kernels can be integrated into SSpMV to enhance its diversity.

IV. Experimental Results

A. Experimental Setup

1) *Machine*: We evaluate SSpMV on two multi-core architectures: an x86_64 platform equipped with two *Intel Xeon Gold 5120* processors, providing a total of 28 CPU cores, and an aarch64 platform with two *Kunpeng 920* processors, offering a total of 128 CPU cores. The MM-Adapter is trained on an NVIDIA-A100 (40GB) GPU.

2) *Dataset*: In addition to the 2,834 real matrices from the SuiteSparse Collection, we generate 3,218 supplementary

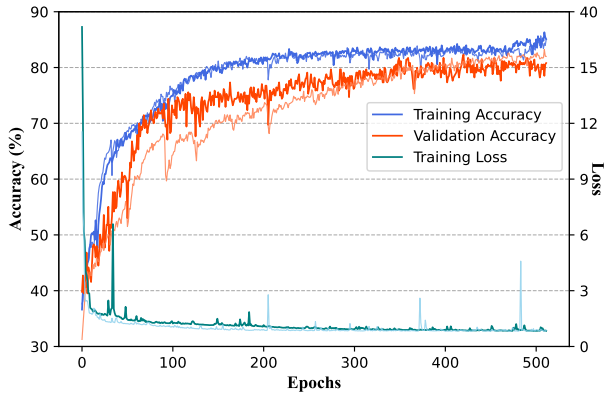


Fig. 7. Loss and accuracy curve of the MM-Adapter.

matrices by sample programs from FreeFEM and OpenFoam, totaling 6,052 matrices. The sample programs include the Poisson equation, Laplace3D equation, Circuit Equation, and other scientific scenarios. The combined dataset is split into 70% for training, 10% for validation, and 20% for testing.

3) *Implementations*: For the source code of various SpMV formats and implementations, we utilize *C++17* and *OpenMP* for shared-memory parallel programming. All algorithms utilize the OpenMP *simd* pragma for automatic vectorization and employ explicit load-balancing optimization strategies.

As summarized in Table I, we use all four scheduling methods for COO and CSR, while DIA utilizes only the *St* scheduling policy. For the S-ELL (ELL) formats, both *StC* and *Dyn* are considered, with chunk size c strictly matching the vector width of the computing platform. The reordering factor σ is chosen from $\{2^9, 2^{14}\}$, aligning with the maximum lengths typically buffered in the L1 and L2 caches, respectively. For BCSR and CSR5, we set $b_R = \sigma_5$ to the cache-line width and $b_C = \omega$ to the vector length, using only the *StC* strategy for chunk parallelism. In total, we obtain 21 candidate implementations for all SpMV formats and parameters.

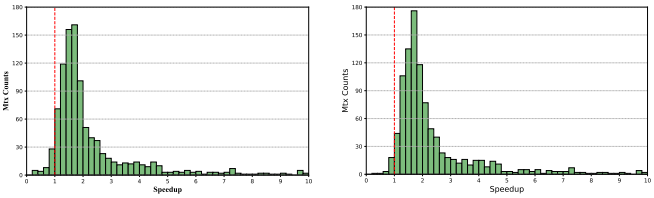
B. MM-Adapter Evaluation

Training the MM-Adapter shown in Fig.6 with 4,236 matrices takes approximately 37 minutes offline. Fig.7 presents the accuracy and loss curves during training and validation, with transparency lines indicating variability over multiple experiments. As training data accumulate, the MM-Adapter progressively refines its modeling accuracy. After 500 epochs, the average training and validation accuracies reach 85.08% and 82.06%, respectively, while the final loss stabilizes around 0.3. The results demonstrate that the MM-Adapter effectively captures the multimodal architectural features relevant to SpMV computation, requiring only a few epochs to converge.

Table II presents the runtime results for 1,210 matrices in the test set and compares MM-Adapter with several existing SpMV prediction models. The results demonstrate significant improvements by MM-Adapter in all three evaluation metrics, achieving 62.63% precision, 59.41% recall, and 81.05% accuracy, respectively. In contrast, the inferior performance of

TABLE II
PREDICTION EVALUATION AND ABLATION STUDY.

Framework	Precision(%)	Recall(%)	Accuracy(%)
SMAT [10]	15.92	13.24	30.69
DIESEL [20]	46.72	48.30	62.77
SpNet [9]	41.21	32.92	75.88
MatNet [21]	52.86	46.76	72.90
MM-Adapter	**62.63**	**59.41**	**81.05**
MM-Adapter Ablation Study			
only channel C1	56.67	51.05	75.94
w/o MLP	60.55	50.47	79.90
w/o Modality 1	56.85	54.72	75.35
w/o Modality 1&3	56.18	51.08	72.63
w/o Modality 2	36.48	31.54	70.15
w/o Expert Features	57.19	51.47	73.51



(a) Predicted algorithms by SSpMV (b) Hand-pick best SpMV algorithms

Fig. 8. Speedup Distribution of isolated SpMV performance over Intel MKL.

SMAT (Decision Tree), DIESEL (MLP), SpNet and MatNet (CNN) underscores their limited generalizability to this multi-classification task, mainly due to their inability to capture sufficient architectural features about SpMV computation.

Moreover, an ablation study is conducted to examine the architecture of MM-Adapter, with results provided in the lower part of Table II. The findings highlight the importance of each component: 1) Different channels enhance the MM-Adapter's capabilities, facilitating the extraction of implicit information in each modality. 2) The MLP module helps the model to be more comprehensive in searching for positive examples, even though its contribution to accuracy is marginal. 3) **M2: Distribution** is identified as the most crucial modality, as it contains the richest structural features, while M1&M3 are similar in importance to the artificial modality.

C. Speedup Distribution

Using the 1,210 matrices in the test set, we evaluate the overall speedup distribution of SSpMV on the Intel Xeon 5120 platform. The SpMV routine from *Intel MKL* is selected as a baseline. As shown in Fig.8a, the acceleration of SSpMV ranges from a minimum of 0.15× to a maximum of 71.7×, with an average speedup of 3.08×. Fig.8b presents the results of hand-picking the best SpMV implementation, where the speedup ranges from 0.21× to 71.7×, with an average of 3.19×. The two figures show similar distributions, with the main difference being the migration of datasets from the [0,1] interval to the [1,2] interval. In addition, some datasets in the [1,2] range failed to achieve optimal acceleration, primarily due to a mismatch in the predicted hyperparameters. In conclusion, with a mere 0.11× performance gap, SSpMV approaches the theoretical optimal algorithm very closely.

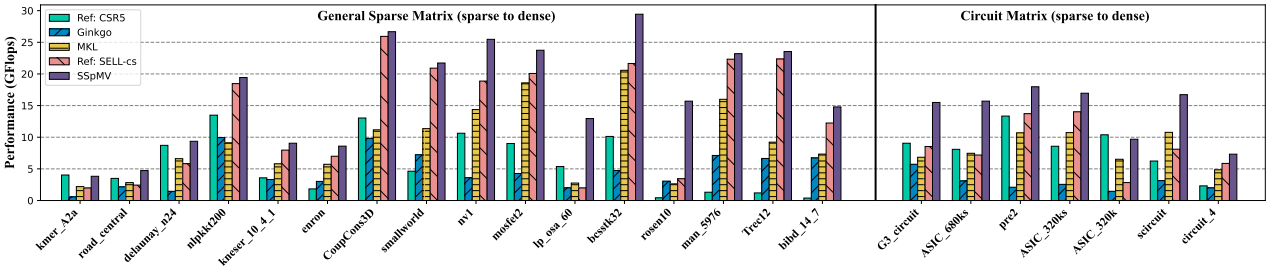


Fig. 9. Performance comparison with other libraries on Intel Xeon 5120 processors.

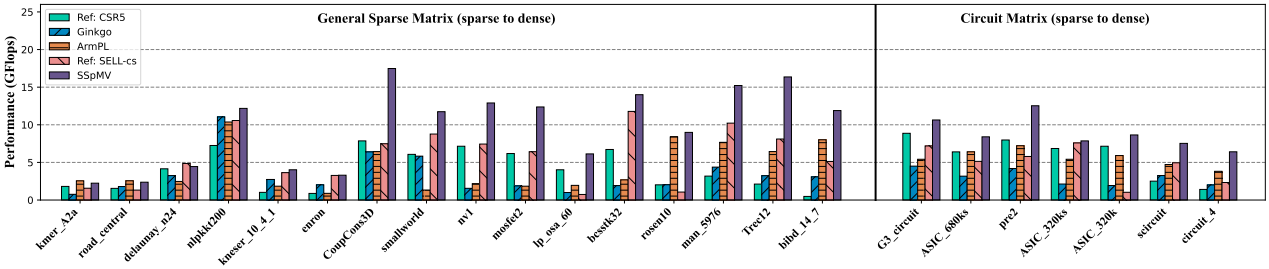


Fig. 10. Performance comparison with other libraries on Huawei Kunpeng 920 processors.

D. Acceleration for Matrix Cases

To assess the performance acceleration of SSpMV, we compare it against vector-friendly formats and well-established sparse linear algebra libraries, including CSR5 [8], SELL-c-σ [7], *Ginkgo* [22], *Intel MKL 24.2.0* [18] on x86_64, and *ArmPL 24.10* [23] on aarch64. In our experiments, CSR5 and SELL-c-σ use their default configuration parameters; while *Ginkgo*, *MKL*, and *ArmPL* are executed by their default executor routines. The SSpMV framework also considers the inference overhead, with an average of 3.19 milliseconds per matrix case. Fig.9 and Fig.10 display double precision SpMV performance results for 16 general sparse matrices and 7 circuit matrices from the SuiteSparse Collection. It is worth noting that circuit matrices represent highly specialized cases due to their irregular distribution of non-zero elements [2].

The results on both platforms show that CSR5 outperforms other formats on extremely sparse matrices due to its compressed tile structure which offers better vector efficiency. In general, *Intel MKL* exhibits the most stable performance across varying matrix types on the x86_64 platform, while both *Ginkgo* and *ArmPL* show significant performance fluctuations on both platforms. This performance gap primarily arises from the structural variations in different sparse patterns, which pose significant challenges in maintaining consistent high performance for SpMV. Fortunately, our SSpMV effectively overcomes this limitation, providing stable and high-performance SpMV across tested matrices. Although inference overhead occasionally impacts speedup, the experimental results demonstrate that SSpMV enhances the adaptability of the SpMV implementations to different sparse patterns.

Overall, on Intel Xeon 5120, compared with CSR5, SELL-c-σ, *Ginkgo* and *Intel MKL*, SSpMV obtains an average speed of $6.70\times$, $1.85\times$, $4.63\times$ and $2.07\times$, respectively. Similarly, on Huawei Kunpeng 920, SSpMV delivers an average speedup of $3.50\times$, $2.51\times$, $3.54\times$ and $2.68\times$ over CSR5, SELL-c-σ, *Ginkgo*, and *ArmPL*. In particular, for matrices from Circuit Simulation Problems, SSpMV can achieve state-of-the-art performance compared with other libraries.

V. Conclusion

In this paper, we implement a suite of high-performance SpMV algorithms tailored to various widely used formats and introduce a Sparsity-aware SpMV framework, SSpMV, which adaptively selects the optimal format and parameters for a given sparse matrix. By leveraging expert-designed features combined with innovative multimodal representations, our MM-Adapter achieves state-of-the-art prediction accuracy. The experimental results show that SSpMV can achieve an average acceleration of $3.08\times$, closely approximating the theoretical optimum of $3.19\times$. Moreover, the matrix case study reveals that SSpMV can deliver significant performance benefits for different matrices across multi-core platforms, demonstrating its effectiveness and portability.

Acknowledgment

We deeply appreciate all the reviewers for their invaluable comments. We also thank Hantao Xiong and Yang He for their significant contributions to this work. This work was partially supported by the Major Projects of Xiangjiang Laboratory (22XJ01011), the Key Program of National Natural Science Foundation of China (U21A20461, U24A20235), the National Natural Science Foundation of China (Grant Nos. 62321003 and 62227808), China Postdoctoral Science Foundation (2024M750879, BX20240111) and Changsha Natural Science Foundation (kq2502277).

REFERENCES

[1] J. Zhao, Y. Wen, Y. Luo, Z. Jin, W. Liu, and Z. Zhou, "Sflu: Synchronization-free sparse lu factorization for fast circuit simulation on gpus," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 37–42.

[2] T. Wang, W. Li, H. Pei, Y. Sun, Z. Jin, and W. Liu, "Accelerating sparse lu factorization with density-aware adaptive matrix multiplication for circuit simulation," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[3] D. Garyfallou, N. Evmorfopoulos, and G. Stamoulis, "Large scale circuit simulation exploiting combinatorial multigrid on massively parallel architectures," in *2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2018, pp. 1–4.

[4] F. Sadi, J. Sweeney, T. M. Low, J. C. Hoe, L. Pileggi, and F. Franchetti, "Efficient spmv operation for large and highly sparse matrices using scalable multi-way merge parallelization," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 347–358.

[5] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

[6] E. F. Kaasschieter, "Preconditioned conjugate gradients for solving singular systems," *Journal of Computational and Applied mathematics*, vol. 24, no. 1-2, pp. 265–275, 1988.

[7] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. R. Bishop, "A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide simd units," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. C401–C423, 2014.

[8] W. Liu and B. Vinter, "Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 339–350.

[9] J. C. Pichel and B. Pateiro-Lopez, "Sparse matrix classification on imbalanced datasets using convolutional neural networks," *IEEE Access*, vol. 7, pp. 82 377–82 389, 2019.

[10] J. Li, G. Tan, M. Chen, and N. Sun, "Smat: An input adaptive auto-tuner for sparse matrix-vector multiplication," in *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, 2013, pp. 117–126.

[11] S. Yesil, A. Heidarshenas, A. Morrison, and J. Torrellas, "Wise: Predicting the performance of sparse matrix vector multiplication with machine learning," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 2023, pp. 329–341.

[12] S. Chen, J. Fang, C. Xu, and Z. Wang, "Adaptive hybrid storage format for sparse matrix–vector multiplication on multi-core simd cpus," *Applied Sciences*, vol. 12, no. 19, p. 9812, 2022.

[13] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Machine learning approach for the predicting performance of spmv on gpu," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2016, pp. 894–901.

[14] Y. Zhao, J. Li, C. Liao, and X. Shen, "Bridging the gap between deep learning and sparse matrix format selection," in *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, 2018, pp. 94–108.

[15] Z. Du, J. Li, Y. Wang, X. Li, G. Tan, and N. Sun, "Alphasparse: Generating high performance spmv codes directly from sparse matrices," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[16] F. Hecht, "New development in freefem++," *Journal of numerical mathematics*, vol. 20, no. 3-4, pp. 251–266, 2012.

[17] H. Jasak, A. Jemcov, Z. Tukovic *et al.*, "Openfoam: A c++ library for complex physics simulations," in *International workshop on coupled methods in numerical dynamics*, vol. 1000, 2007, pp. 1–20.

[18] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, "Intel math kernel library," *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures*, pp. 167–188, 2014.

[19] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[20] T. Mohammed, A. Albeshri, I. Katib, and R. Mehmood, "Diesel: A novel deep learning-based tool for spmv computations and solving sparse linear equation systems," *The Journal of Supercomputing*, vol. 77, no. 6, pp. 6313–6355, 2021.

[21] Z. Xie, G. Tan, W. Liu, and N. Sun, "Ia-spgemm: An input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 94–105.

[22] H. Anzt, T. Cojean, G. Flegar, F. Göbel, T. Grützmacher, P. Nayak, T. Ribizel, Y. M. Tsai, and E. S. Quintana-Ortí, "Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing," *ACM Transactions on Mathematical Software*, vol. 48, no. 1, pp. 2:1–2:33, Feb. 2022. [Online]. Available: https://doi.org/10.1145/3480935

[23] Arm, "Arm performance libraries," https://developer.arm.com/tools-and-software/server-and-hpc/compile/arm-compiler-for-linux/arm-performance-libraries.

Comparison-Free Bit-Stream Generation for Cost-Efficient Unary Computing

Faeze S. Banitaba*, Amir Hossein Jalilvand*, M. Hassan Najafi†, Sercan Aygun*
*University of Louisiana at Lafayette, †Case Western Reserve University
*{faeze.banitaba, amir.jalilvand, sercan.aygun}@louisiana.edu, †mhassan.najafi@case.edu

***Abstract*—Today, unconventional hardware design techniques based on simple data representations are receiving more and more attention. Unary computing is one of these techniques that processes data in the form of uniform bit-streams. The simplicity of implementing complex arithmetic operations and high tolerance to noise are the crucial advantages of unary systems. However, converting data from weighed binary radix to unary representation with existing *comparator-based* unary number generators is expensive regarding footprint area and power consumption. The problem aggregates when the number of inputs and data precision increase. This work proposes a low-cost, comparison-free, unary number generation mechanism for efficient data conversion from binary radix to unary representation. We introduce a serial and two parallel (an exact and an approximate) unary number generators. Synthesis results show that the proposed method reduces the hardware area, power consumption, and area-delay product for both serial and parallel designs compared to the state-of-the-art converter. We evaluate the efficiency of the proposed converter in four use cases.**

I. Introduction

Designers encounter stringent design constraints in modern digital hardware systems, particularly regarding area footprint and power consumption [1]. In digital embedded systems, the primary considerations are optimizing power consumption and minimizing the physical area, as these factors critically affect the overall cost and energy efficiency of the systems. Traditional digital systems predominantly utilize weighted binary radix-2 for data representation. Computations based on this positional representation are complex and costly to implement [2]. Due to the complexity and high cost associated with conventional binary designs, unconventional design techniques that leverage simpler data representations are gaining increasing attention [3].

Unary computing (UC) [4] is an unconventional technique that processes data in the form of uniform bit-streams with the ability to produce deterministic and accurate results [5], [6]. Input values are encoded in the unary domain through a sequence of one value ('0' or '1') followed by a sequence of the other value ('1' or '0', respectively). For example, 11000 is a left-aligned unary bit-stream representing $2/5$. Depending on the location of '1's in the stream, unary bit-streams can be either right- or left-aligned. The value of a unary bit-stream is determined by calculating the ratio of the number of 1s to the total length of the bit-stream, indicating the density or probability of observing 1. While data is typically confined to the [0, 1] interval, values outside this range can be presented by scaling. A key difference from binary radix representation is that all digits in a unary bit-stream have equal weight. This provides immunity to noise, as any bit flip results in only a least significant bit (LSB) error in the represented value.

The primary advantage of UC is its simplicity in implementing complex arithmetic operations, which enables ultra-low-cost hardware designs [5]. For instance, multiplication operation can be realized by bit-wise `AND` operation on *uncorrelated* unary bit-streams [7]. Moreover, minimum (`Min`), maximum (`Max`), and absolute difference operations can be implemented by bit-wise `AND`, `OR`, and `XOR` operations, respectively, on *correlated* unary bit-streams [5]. Fig. 1(a) shows the logic design of the `Max` and `Min` value functions in the weighted binary and unary domains. As can be seen, the complexity and implementation cost of the binary design depends on the data precision (m) requiring an m-bit binary comparator and two m-bit multiplexers (`MUX`s). This results in a higher hardware area and power cost than the unary counterpart implemented using only one `AND` and one `OR` gate (Fig. 1(b)). The unary implementation is independent of the data precision. The same design can process data with higher precision by processing longer bit-streams. UC has been used for the hardware-efficient implementation of different applications, from fuzzy logic [8] to hyperdimensional computing [9], image processing [10], deep neural network architecture [11], sorting [12], and neuromorphic computing [13], [14].

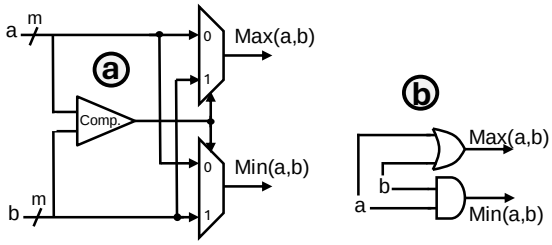


Fig. 1. Implementation of the `Min` and `Max` value functions; (a) Conventional binary design (the design cost depends on the data size or m), (b) Unary design (the design cost is independent of the bit-stream length).

An essential step for processing data in unary systems is to convert data from conventional binary to unary bit-streams. A unary bit-stream of 2^m bits is required to accurately represent a data value with a resolution of 2^{-m} [5]. This means a unary bit-stream of 256 bits is needed to accurately represent an 8-bit precision number. Fig. 2(a) illustrates the common structure of a unary bit-stream generator built from an m-bit counter, an m-bit binary comparator, and an m-bit register. The output of comparing the counter and the register values produces one bit of the bit-stream in each clock cycle. A parallel unary bit-stream generator (with degree 2 of parallelism) is also shown in Fig. 2(b). In each clock cycle, the counter output is added by 1 bit ('0' or '1') and compared with the register. Two bits of the unary bit-stream are generated in each cycle with this parallel generator. This structure is costly compared to the computation logic of the unary circuits, consuming over 80% of the unary system's total area and power cost [5].

Stochastic computing (SC) [15] is another emerging computing paradigm that shares many properties with UC but is known for approximate computations and operation on *random* bit-streams. SC uses binary random number generators instead of binary counters for

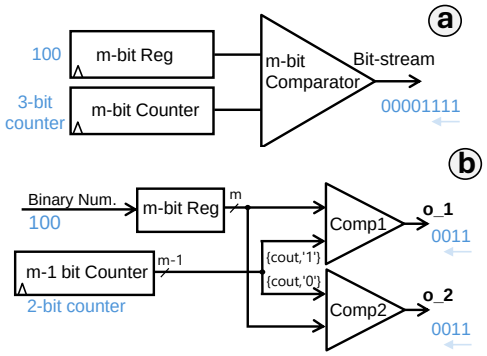


Fig. 2. Conventional comparator-based unary bit-stream generator: (a) A serial design [5], (b) A 2× parallel design.

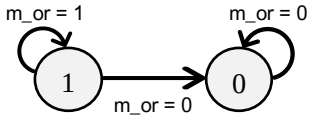


Fig. 3. FSM of the proposed bit-stream generator for left-aligned unary bit-streams, with '1's preceding '0's.

bit-stream generation. While early stochastic bit-stream generators were built using a similar comparator-based structure, finite-state-machine (FSM)-based bit-stream generators have been introduced to reduce the data conversion cost of SC systems. Asadi *et al.* [16] proposed a low-cost FSM-based low-discrepancy (LD) bit-stream generator that supports the generation of quasi-random bit-streams with recurrent LD distributions. Their proposed generator relies on an FSM and a MUX unit for low-cost bit-stream generation. However, their method only supports random bit-streams and cannot generate unary bit-streams.

This work proposes a low-cost, comparison-free unary bit-stream generator that can efficiently convert data from binary radix to unary representation. Synthesis results show up to 85% and 92% savings in the area and power consumption, respectively, compared to those of the conventional comparator-based unary bit-stream generator when converting a single input. When converting eight binary inputs to their corresponding unary bit-streams, up to 62% and 58% savings in the area and power are obtained. But processing fully serial unary bit-streams can result in extended processing latency and high energy consumption. To address this issue, we propose two parallel unary bit-stream generators: 1) an approximate and 2) an accurate parallel bit-stream generator. The accurate design generates unary bit-streams accurately (zero conversion error) at some additional costs compared to the approximate one.

The rest of the paper is organized as follows: Section II describes the proposed unary bit-stream generators. Section III compares the hardware costs of the proposed bit-stream generator to those of the conventional comparator-based generator. Section IV evaluates the proposed bit-stream generator in several use cases. Finally, Section V concludes the paper.

II. Proposed Architecture

This section discusses the architecture of the proposed bit-stream generator. The proposed generator converts an m-bit precision binary input data to a right-aligned unary bit-stream. The proposed converter is based on a two-state FSM, illustrated in Fig. 3. As can be seen,

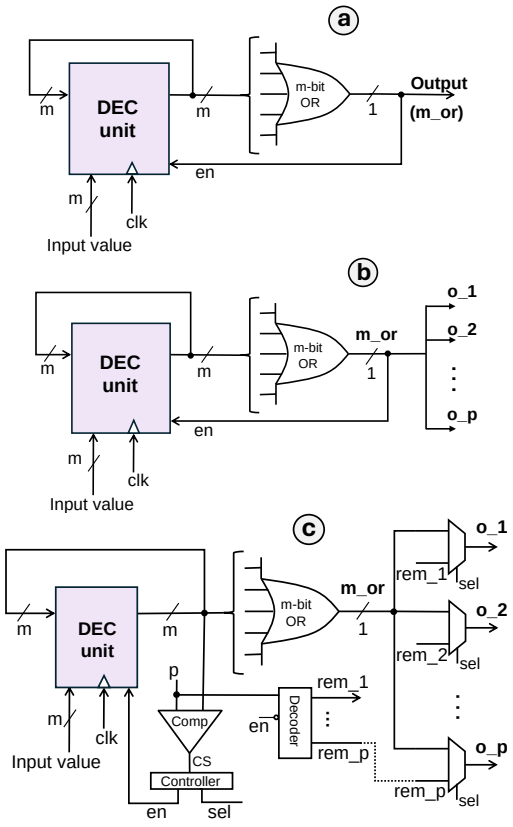


Fig. 4. Proposed designs for unary bit-stream generation: (a) Serial architecture, (b) Approximate Parallel design, (c) Accurate Parallel design.

the FSM has only one transition: when the FSM is in state '1', the output is '1', which then becomes '0' when the state changes to '0'. When the *m_or* signal becomes '0', the FSM goes from state '1' to state '0'. In what follows, we discuss a serial and two parallel architectures for the proposed bit-stream generator.

A. Serial Design

Fig. 4(a) shows the serial architecture of the proposed bit-stream generator. The core of this bit-stream generator is a decrement (DEC) unit and an m-input OR gate. Consider an input value of 0.5, equivalent to '0.100' in the binary domain with a data precision of $m = 3$. The bit-stream generator converts this binary input to an 8-bit unary bit-stream, i.e., '00001111'. Table I shows a step-by-step example of bit-stream generation with this design. In the proposed design, an input register is initialized by the input value (here, '0.100'). In the first cycle, the enable input of the DEC unit is '0', so the input is not decremented and passes as is. The input enters the m-bit OR gate and re-enters the DEC unit again in the next clock cycle. Now, the output of the OR gate becomes '1'. The same input goes to the DEC unit in the second cycle, but the enable signal is '1' this time. Now, the input is subtracted by *one unit* and then entered the OR gate. The process continues until the result from the OR gate equals 0, which, as a result, disables the DEC unit. In this state, *m_or*=0, the FSM state changes from '1' to '0', and the output remains on '0' for the remaining cycles. In the example here, with the input equal to '0.100', the output signal remains '1' until exactly four '1's are generated at the output. When the 4th '1' is returned to the DEC unit (as enable signal), the output from the DEC unit becomes '0', which also makes the output of the OR gate '0'.

TABLE I
A STEP-BY-STEP EXAMPLE OF BIT-STREAM GENERATION USING THE PROPOSED SERIAL DESIGN (INPUT VALUE=100, $m = 3$)

clk	DEC			m_or (Output Bit)
	in	out	en	
0	100	100	0	1
1	100	011	1	1
2	011	010	1	1
3	010	001	1	1
4	001	000	1	0
5	000	000	0	0
6	000	000	0	0
7	000	000	0	0

B. Parallel Design

Serial UC suffers from long latency as the number of processing cycles increases exponentially with the data precision. For a data precision equivalent to m binary bits, one needs to process unary bit-streams of 2^m bits in 2^m cycles. Given the relatively small area of unary logic compared to conventional binary designs, we can execute unary computations in parallel to reduce latency. To this end, we propose two designs for parallel generation of unary bit-streams: 1) an *approximate* design and 2) an *accurate* design. The approximate design offers lower hardware cost at the expense of accuracy, while the accurate design eliminates errors at a higher hardware cost. We define the degree of parallelism as p where $1 < p \leq m$.

1) Approximate Design: In the first design, the bit-stream generator is similar to the circuit shown in Fig. 4(a) but with some small modifications. In this design, the number of output ports and the value by which the DEC unit subtracts the input are both equal to the degree of parallelism or p. For example, for $p = 2$, the DEC unit subtracts the input by '0.010' or '0.000', depending on the enable input. Fig. 4(b) shows the proposed design, and Table II demonstrates a step-by-step example. We denote the output ports with o_1, ..., o_p. Assume that the input binary value is '0.100' and $m = 3$. In this case, the DEC unit subtracts the input value by '0.000' in the first cycle. The output of subtraction is '0.100', which goes back to the input and OR gate. The *m_or* signal becomes '1' and drives all the outputs (o_1, ..., o_p). At the same time, a '1' is returned to the enable input of the DEC unit. Next, the input ('0.100') is subtracted by '0.010', which produces '0.010'. The *m_or* signal remains on '1'. The output stream becomes $1, 1, 1$. The input ('0.010') is subtracted again by '0.010', which produces '0.000' this time. Now the *m_or* signal becomes '0', which results in '0' for all output ports (the output stream becomes $0, 0, 0$). Similar to the serial bit-stream generator, the FSM state changes, and the remaining output bits become '0'.

We note that if the number of '1's in the output ports is a multiple of p (e.g., four '1's in the example), this method produces an accurate output bit-stream. However, if the above condition is not met, there is an error in the generated bit-streams. For example, if the input value is '0.101', p is 2, and m is 3, the generated bit-stream contains four '1's, which is one bit of '1' short compared to the desired output bit-stream with five '1's. Table III reports the accuracy of the first parallel design in terms of mean absolute error (MAE) for converting data to unary bit-streams for different values of m and p.

2) Accurate Design: In the second method, we make some modifications to the first parallel design to accurately generate sub-bit-streams. Fig. 4(c) shows the proposed design, and Table II a step-by-step example of this method. This design includes a comparator, a decoder, p 2-to-1 MUXs, and a controller. In the first cycle, the select

TABLE II
A STEP-BY-STEP EXAMPLE OF BIT-STREAM GENERATION BY OUR PROPOSED PARALLEL DESIGNS (INPUT VALUE=100, $m = 3$ AND $p = 2$)

clk	Approximate design						Accurate design								
	DEC			m_or	o_1	o_2	DEC			m_or	rem_1	rem_2	o_1	o_2	CS
	in	out	en				in	out	en						
0	100	100	0	1	1	1	100	100	0	1	-	-	1	1	0
1	100	010	1	1	1	1	100	010	1	1	-	-	1	1	0
2	010	000	1	0	0	0	010	000	1	0	0	0	0	0	1
3	000	000	0	0	0	0	000	000	0	0	0	0	0	0	1

TABLE III
MAE (%) OF THE FIRST PARALLEL GENERATOR FOR DIFFERENT DEGREES OF PARALLELISM (2×, 4×, 8×)

Input Precision	First Parallel Generator		
	2×	4×	8×
$m = 5$	1.56	4.68	10.93
$m = 6$	0.78	2.34	5.46
$m = 7$	0.39	1.17	2.73
$m = 8$	0.19	0.58	1.36
$m = 9$	0.09	0.29	0.68
$m = 10$	0.04	0.14	0.34

input (sel) of the MUXs is '0'. In each clock cycle, the comparator compares the output of the DEC unit with p. When comparator output (*CS*) is '1', it means that there is a remainder less than p, and we need to change the controller's state (shown in Fig. 5) from *Normal* to *RemOut*. In the *RemOut* state, the controller sets $en = 0$ and $sel = 1$. When the sel signal becomes one, the output bit-streams will be fed with the output of the decoder ($rem_1 \ldots rem_p$). The output of the decoder shows how many output bits must be set to '1'. For example, with $p = 4$, if the remainder is '0.011', we need to set o_1, o_2, and o_3 to '1' by encoding the remainder value ($rem_1 \ldots rem_4$) to '1110'. This parallel design converts the input value to p sub-bit-streams accurately in parallel.

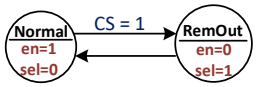


Fig. 5. The controller for accurate parallel design.

III. HARDWARE COST

In this section, we compare the hardware costs of the proposed serial and parallel unary bit-stream generators with that of the conventional comparator-based unary bit-stream generator commonly used in UC systems.

A. Serial Design

We developed an RTL VHDL description for the proposed comparison-free and the conventional comparator-based unary bit-stream generator. We synthesized the designs using the Synopsys Design Compiler v2018.06 with a 45nm FreePDK library [17] and also with Xilinx Vivado v2018.2 on a Virtex-7 FPGA. We report the synthesis results for different data bit-widths (i.e., $m = 4, 5, 6, 7$, and 8) and two cases of converting *one input* and *eight inputs* to unary bit-streams. When converting multiple inputs, the counter can be shared in the conventional design between several bit-stream generators to save the hardware cost. Table IV reports the synthesis results in terms of hardware area, power consumption, critical path latency (CP), and the number of occupied lookup tables (LUTs). As can be seen, the proposed (serial) generator achieves lower values

TABLE IV

SYNTHESIS RESULTS OF THE IMPLEMENTED UNARY BIT-STREAM GENERATORS (PROPOSED SERIAL VS. CONVENTIONAL COMPARATOR-BASED) FOR CONVERTING DIFFERENT NUMBERS OF INPUTS (N) AND DATA PRECISIONS (m).

N	m	Area (μm^2)		Power @$1GHz$ (μW)		CP (ns)		Occupied LUT	
		Proposed	Conventional	**Proposed**	Conventional	**Proposed**	Conventional	**Proposed**	Conventional
1	4	**19.2**	114	**17.9**	167	**0.21**	0.46	**1**	4
	5	**24.4**	142	**17.6**	195	**0.24**	0.48	**2**	5
	6	**25.3**	169	**18.5**	218	**0.27**	0.52	**2**	8
	7	**27.6**	210	**19.6**	246	**0.31**	0.56	**3**	9
	8	**35.3**	237	**20.7**	267	**0.38**	0.61	**3**	10
8	4	**153**	265	**140**	231	**0.21**	0.46	**8**	14
	5	**195**	405	**141**	290	**0.24**	0.48	**16**	16
	6	**202**	530	**148**	331	**0.27**	0.52	**16**	24
	7	**221**	589	**162**	359	**0.31**	0.56	**24**	37
	8	**282**	760	**165**	394	**0.38**	0.61	**24**	43

TABLE V

HARDWARE AREA (μm^2), CRITICAL PATH LATENCY (ns), AND AREA DELAY PRODUCTS (A×D) OF THE COMPARATOR-BASED, THE PROPOSED APPROXIMATE AND ACCURATE DESIGN FOR THE PARALLEL GENERATION OF UNARY BIT-STREAMS FOR 8-BIT PRECISION DATA.

Parallel Design	1×			2×			4×			8×		
	Area	CP	$A \times D$	Area	CP	$A \times D$	Area	CP	$A \times D$	Area	CP	$A \times D$
Comparator-based	237	0.61	144	451	0.61	275	901	0.61	550	1414	0.61	862
Approximate design	35	0.38	13.4	41	0.38	15	43	0.38	16	47	0.38	17
Exact design	35	0.38	13.4	52	0.40	21	70	0.41	28	128	0.45	57

than the conventional approach in all metrics. For instance, in the case of converting one input when $m = 8$, the circuit area of the proposed design is 35.3 μm^2 compared to 237 μm^2 of the comparator-based generator. This shows 85% saving in area. For the same case, the power consumption of the proposed design is 20.7 μW, providing 92% saving compared to 267 μW of the prior design. In the case of converting eight inputs when $m = 8$, the circuit area of the proposed design is 282 μm^2 compared to 760 μm^2 of the conventional generator, which shows 62% saving in area. For this case, the power consumption of the proposed approach is 165μW, 58% lower compared to 394 μW of the comparator-based generator.

When increasing the number of inputs from one to eight, the proposed design still saves the hardware area and power cost considerably compared to the comparator-based generator. A higher saving rate (i.e., 85%) was observed for the case of converting a single input because the dominant cost in the conventional generator was the cost of the counter. However, when converting multiple inputs, the counter is shared between several generators, and the dominant is the cost of a new binary comparator for each additional input. Considering the lower cost of the proposed m-bit design compared to that of an m-bit binary comparator (35 μm^2 for the proposed vs. 73 μm^2 for the comparator when m=8), the proposed design shows scalability, and similar cost savings are expected when increasing the number of inputs. As it can be seen in Table IV, in terms of CP, the proposed design provides a lower latency, which means for the same number of processing cycles it works faster and reduces the total latency. For example, for $m = 4$, our design reduces CP by 54% from 0.46 ns to 0.21 ns, and for $m = 8$ by 38% from 0.61 ns to 0.38 ns.

B. Parallel Design

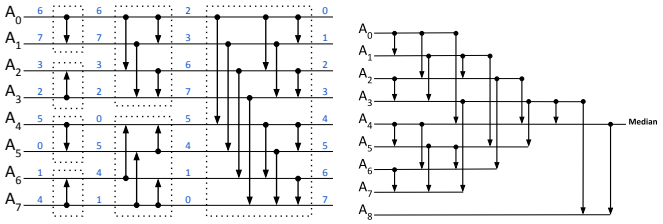
Table V reports the hardware area, CP, and area-delay products of the conventional comparator-based, and the proposed approximate and accurate parallel designs for 2×, 4×, and 8× degrees of parallelism. We report the hardware costs for processing 8-bit precision data. We also report the cost of non-parallel (1×) unary bit-stream generation in this table. As reported, for the case of 2× parallel design, the hardware area costs of the proposed approximate and accurate parallel designs are 41.3 μm^2 and 52.5 μm^2, respectively, while the comparator-based design (Fig. 2(b)) occupies 450 μm^2. The area-delay product of the approximate and accurate parallel designs are 15 and 21, respectively, while for the comparator-based design it is 275, 13× greater than the accurate parallel design. In the case of 4× parallel design, the area-delay product of the accurate design is 28, while for the comparator-based unary generator, it is 550. Finally, for 8× parallel design, the area-delay product of the comparator-based design is 15× and 50× more than that of the accurate and approximate parallel design, respectively. Evidently, the proposed design significantly reduces the area, latency, and area-delay product of parallel unary bit-stream generation compared to the conventional comparator-based unary bit-stream generator.

IV. CASE STUDY

Next, we evaluate the hardware efficiency of the proposed bit-stream generator for four state-of-the-art UC systems for 1) Sorting, 2) Median filtering, 3) Robert's edge detection, and 4) Gamma correction.

A. Sorting Networks

Sorting is an important task in different applications from databases to artificial intelligence, scientific computing, and image processing. For applications that require high performance, sorting is often performed in hardware. A common approach for hardware sorting is to wire up a network of compare-and-swap (CAS) units. The incoming data is sorted as it passes the network. Each CAS block compares two input values and swaps the values at the output if required. There are two variants of CAS blocks: an "ascending (↑)" type and a "descending (↓)" type. Fig. 6 shows the CAS network for sorting eight input data.



(a) (b)

Fig. 6. The CAS network for (a) sorting 8 input data, and (b) a 3x3 median filter circuit [5].

The authors in [5] proposed an area- and power-efficient implementation for sorting networks based on UC. In their design, each CAS unit is implemented using one AND and one OR gate. Their design uses the conventional comparator-based design for converting data from binary to unary representation. To improve the sorting designs of [5], we replace the conventional unary bit-stream generator used in their designs with our proposed design. We compare the hardware cost for three cases of 1) the conventional implementation with weighted binary data (WB), 2) UC design with comparator-based unary bit-stream generator [5], and 3) UC design with the proposed converter. Table VI compares the hardware area and power costs in 45nm technology. The number of input data (i.e., sorting network size) ranges from 8 to 256 with data-width $m = 8$. As it can be seen, in the case of implementing 8-input sorting network (shown in Fig. 6), the sorting design with the proposed bit-stream generator occupies $2,242\ \mu m^2$, while the design with comparator-based unary generator, and the weighted binary implementation consume $2,706\ \mu m^2$ and $3,086\ \mu m^2$ area, respectively. A similar saving is observed for the power consumption. The area and power savings with the UC designs increase as the sorting network size increases. Our proposed converter reduces the area and power consumption of the sorting circuits up to 84% and 93% compared to the conventional weighted binary and 17% and 36% compared to the comparator-based UC sorting.

We also implemented the parallel unary design of the sorting networks with the comparator-based and also with the proposed accurate parallel design with degrees of $2\times$, $4\times$, and $8\times$. Table VII reports the hardware area costs. We only compare the area footprint in this case as the power consumption is correlated with the area and a similar improvement is expected. The synthesis results show the superiority of the sorting circuit implemented with the proposed parallel generator over the sorting design with parallel comparator-based unary generator. For example, in the case of sorting 256 input data with the $8\times$ parallel design, the design with our accurate converter occupies $164,116\ \mu m^2$ while the one with comparator-based generator consumes $396,594\ \mu m^2$ area. This means 58.6% area saving with the proposed parallel unary generator.

B. Median Filtering

A median filter is a popular nonlinear filter widely used in image, speech, and signal processing applications [18], [19], [20]. It replaces each input data with the median of all the data in a local neighborhood. This results in filtering out impulse noise and smoothing the image while preserving important properties such as edge information. Sorting network-based median filter consisting of a network of CAS blocks is a common approach for the hardware design of median filter. The authors in [5] proposed a low-cost UC design for median filtering using CAS networks. We implemented and synthesized a 3×3 UC-based median filter design with both the comparator-based and the proposed unary generator. The hardware area costs are reported in Table IX. As it can be seen, our approximate and accurate parallel designs reduce the area cost of the state-of-the-art UC median filter [5] up to 78% and 69%, respectively. We further evaluate the performance of the UC design with the proposed parallel designs for different bit-stream lengths for 1000 sets of random input values. Table VIII reports the MAE results. As can be seen, the accurate designs achieve 0.00% MAE with a bit-stream length of 256. For the approximate parallel designs, increasing the degree of parallelism increases the MAE for a fixed bit-stream length. However, by increasing the bit-stream length, the error rate decreases quickly.

TABLE VI
SYNTHESIS RESULTS OF SORTING CIRCUITS WITH DIFFERENT NUMBERS OF INPUTS WITH CONVENTIONAL WEIGHTED BINARY (WB), COMPARATOR-BASED UC [5], AND PROPOSED DESIGN FOR $m = 8$.

Input	Area (μm^2)			Power @ $50MHz$ (μW)		
	WB	[5]	**Proposed**	WB	[5]	**Proposed**
8	3,086	2,706	**2,242**	120	130	**83**
16	10,534	5,444	**4,680**	490	250	**170**
32	32,508	11,196	**9,833**	1,750	490	**360**
64	90,691	23,279	**20,719**	5,480	960	**750**
128	242,049	49,756	**44,799**	15,480	1,800	**1450**
256	586,456	106,037	**96,288**	42,200	3,640	**2960**

TABLE VII
HARDWARE AREA COST OF UC SORTING CIRCUITS WITH DIFFERENT NUMBERS OF INPUTS FOR $2\times$, $4\times$, AND $8\times$ **PARALLEL** BIT-STREAM GENERATION WITH THE COMPARATOR-BASED [5] AND WITH OUR ACCURATE PARALLEL UNARY GENERATOR FOR $m = 8$.

Input	$2\times$		$4\times$		$8\times$	
	[5]	**Proposed**	[5]	**Proposed**	[5]	**Proposed**
8	2,266	**1,193**	4,021	**1,597**	7,531	**2,059**
16	4,668	**2,780**	8,171	**3,983**	15,321	**4,907**
32	9,951	**6,504**	17,572	**9,855**	33,102	**11,702**
64	21,678	**15,113**	38,696	**23,920**	73,307	**27,616**
128	49,750	**36,948**	90,179	**61,286**	172,190	**68,678**
256	112,551	**87,277**	206,461	**149,334**	396,584	**164,116**

C. Robert's cross edge detection

We further implement Robert's cross edge detector circuit as proposed in [21]. Each Robert's cross operator consists of a pair of 2 × 2 convolution kernels that process an image pixel based on its three neighbors as follows:

$$S_{i,j} = 0.5 \times (|r_{i,j} - r_{i+1,j+1}| + |r_{i,j+1} - r_{i+1,j}|)$$

where $r_{i,j}$ is the value of the pixel at location (i, j) of the input image and $S_{i,j}$ is the output value computed for the same location in the output image. The circuit is shown in Fig. 7(a). In this design, two XOR gates are used to implement the absolute value subtractions and one MUX unit is used for the scaled addition. The four inputs to the XOR gates must be correlated bit-streams, but they should all be uncorrelated to the select input of the MUX, which is a bit-stream representing 0.5. For converting the four main inputs from binary to unary, we use the proposed parallel design. We evaluated the performance of both approximate and accurate parallel designs when used for converting the inputs of Robert's cross circuit. Table VIII reports the accuracy results for different bit-stream lengths and degrees of parallelism and Table IX compares the hardware area costs. As can be seen in the reported numbers, the proposed designs achieve a high accuracy for most cases. Especially with bit-stream

TABLE VIII

MAEs OF THREE APPLICATION CASE STUDIES FOR DIFFERENT DEGREES OF PARALLELISM (1×, 2×, 4×, 8×) AND BIT-STREAM LENGTHS FOR THE PROPOSED PARALLEL DESIGNS. THE INPUT DATA BIT-WIDTH (m) IS 8.

Design	Bit-stream Length	1×		2×		4×		8×	
		Accurate	Approximate	Accurate	Approximate	Accurate	Approximate	Accurate	Approximate
3×3 Median Filter	8	1.84	1.78	1.81	4.19	1.82	12.81	1.78	43.79
	16	1.02	0.99	0.95	2.13	1.01	5.46	1.00	14.14
	32	0.59	0.58	0.59	1.12	0.60	2.70	0.60	6.21
	64	0.39	0.39	0.38	0.63	0.39	1.37	0.39	3.33
	128	0.29	0.29	0.30	0.39	0.30	0.70	0.31	1.52
	256	0.00	0.00	0.00	0.30	0.00	0.88	0.00	2.04
	512	0.00	0.00	0.00	0.00	0.00	0.29	0.00	0.90
Robert's Cross Edge Detection	8	4.64	4.00	4.26	4.87	4.23	11.34	4.39	16.81
	16	2.16	2.08	2.11	2.62	2.15	5.95	2.00	12.33
	32	1.03	0.99	1.03	1.29	1.03	3.11	1.09	6.52
	64	0.54	0.54	0.55	0.69	0.54	1.56	0.53	3.37
	128	0.28	0.27	0.25	0.38	0.26	0.85	0.26	1.82
	256	0.11	0.10	0.12	0.13	0.10	0.39	0.11	0.84
	512	0.00	0.00	0.00	0.00	0.00	0.12	0.00	0.38
Gamma Correction	8	11.09	11.81	11.04	11.36	11.22	16.22	11.19	30.06
	16	7.74	7.64	7.95	8.05	8.38	8.89	8.10	15.64
	32	4.84	4.91	5.09	5.27	5.20	7.57	5.30	11.21
	64	3.52	3.43	3.55	3.63	3.51	3.96	3.69	6.97
	128	2.53	2.49	2.54	2.64	2.48	2.52	2.40	3.08
	256	1.76	1.72	1.75	1.85	1.74	1.86	1.72	1.89
	512	1.28	1.29	1.27	1.35	1.24	1.29	1.31	1.34

TABLE IX

HARDWARE AREA COST OF UC CIRCUITS FOR 2×, 4×, AND 8× PARALLEL BIT-STREAM GENERATION WITH COMPARATOR-BASED (CB), AND WITH PROPOSED PARALLEL DESIGNS FOR $m = 8$

Design	2×			4×			8×		
	CB	Approx.	Accurate	CB	Approx.	Accurate	CB	Approx.	Accurate
Median	2,349	1,104	1,205	4,131	1,280	1,524	7,695	1,632	2,359
Robert	1,175	439	484	2,098	466	574	3,943	520	843
Gamma	2,138	995	1,085	3,772	1,165	1,381	7,040	1,505	2,151

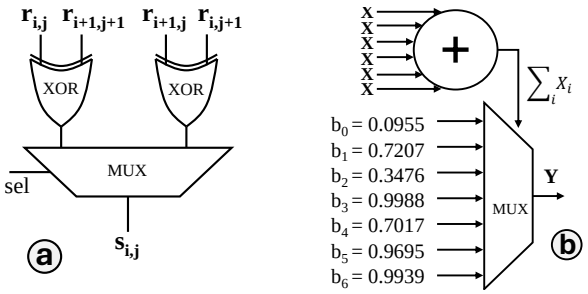


Fig. 7. UC designs for (a) Robert's cross edge detection [21], (b) Gamma correction function [15].

lengths of more than 32 bits, they achieve a MAE of less than 1%. The synthesis results show up to 86% area savings with proposed design compared the conventional comparator-based design.

D. Gamma Correction

Finally, we implement the Gamma correction circuit proposed in [15] and shown in Fig. 7(b). This circuit implements the Gamma correction function ($f(x) = x^{0.45}$) by approximating the function using a Bernstein polynomial and with a MUX-based architecture, called ReSC [15]. The inputs to this circuit are six independent bit-streams, each with a probability corresponding to the value x and seven correlated unary bit-streams, set to constant values corresponding to the Bernstein coefficients. We implemented the design with both comparator-based and proposed unary converters. Table IX compares the area cost numbers. We did not consider the cost of generating independent bit-streams in our area comparison as it's the same in both UC design approaches. As can be seen, the proposed design reduces the hardware area cost of the UC Gamma correction circuit by up to 78%. Table VIII further reports the accuracy results with both accurate and approximate parallel design. A higher error rate compared to the Robert and Median designs is due to the approximation of the Gamma correction function with the ReSC architecture.

V. CONCLUSION

In this work, we proposed a novel comparison-free unary bit-stream generation approach for the cost-efficient conversion of data from binary to unary bit-streams. We proposed a serial and two parallel (an *accurate* and an *approximate*) unary bit-stream generators. We evaluated the hardware cost of the proposed designs compared to the conventional comparator-based binary-to-unary converter. We further evaluated the accuracy performance for different bit-stream lengths and degrees of parallelism. Synthesis results showed up to 85% and 92% savings in the area and power costs, respectively, compared to the state-of-the-art comparator-based unary bit-stream generator.

The MATLAB and synthesizable HDL source codes associated with this work are publicly available at: https://github.com/faeze-bnt/Bit-stream-Gen.

VI. ACKNOWLEDGEMENT

This work is supported in part by the National Science Foundation under grants #2019511, #2339701, and a generous gift from NVIDIA.

REFERENCES

[1] S. S. Gill, H. Wu, P. Patros, C. Ottaviani, P. Arora, V. C. Pujol, D. Haunschild, A. K. Parlikad, O. Cetinkaya, H. Lutfiyya *et al.*, "Modern computing: Vision and challenges," *Telematics and Informatics Reports*, p. 100116, 2024.

[2] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. San Miguel, "Ugemm: Unary computing architecture for gemm applications," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 377–390.

[3] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1515–1531, 2017.

[4] W. J. Poppelbaum, A. Dollas, J. Glickman, and C. O'Toole, "Unary processing," in *Advances in computers*. Elsevier, 1987, vol. 26, pp. 47–92.

[5] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-cost sorting network circuits using unary processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 8, pp. 1471–1480, 2018.

[6] S. R. Faraji and K. Bazargan, "Hybrid binary-unary hardware accelerator," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 210–215.

[7] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2925–2938, 2019.

[8] A. H. Jalilvand, M. H. Najafi, and M. Fazeli, "Fuzzy-logic using unary bit-stream processing," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[9] M. Shoushtari Moghadam, S. Aygun, F. S. Banitaba, and M. H. Najafi, "All you need is unary: End-to-end unary bit-stream processing in hyperdimensional computing," in *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, 2024, pp. 1–6.

[10] A. Khataei, G. Singh, and K. Bazargan, "Approximate hybrid binary-unary computing with applications in bert language model and image processing," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2023, pp. 165–175.

[11] D. Wu, J. Li, Z. Pan, Y. Kim, and J. S. Miguel, "ubrain: A unary brain computer interface," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 468–481.

[12] A. H. Jalilvand, F. S. Banitaba, S. N. Estiri, S. Aygun, and M. H. Najafi, "Sorting it out in hardware: A state-of-the-art survey," *ACM Transactions on Design Automation of Electronic Systems*, 2025.

[13] C. Ma, Y. Sun, W. Qian, Z. Meng, R. Yang, and L. Jiang, "Go unary: A novel synapse coding and mapping scheme for reliable reram-based neuromorphic computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1432–1437.

[14] J. Smith, "Space-time algebra: A model for neocortical computation," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 289–300.

[15] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE transactions on computers*, vol. 60, no. 1, pp. 93–105, 2010.

[16] S. Asadi, M. H. Najafi, and M. Imani, "A low-cost fsm-based bit-stream generator for low-discrepancy stochastic computing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 908–913.

[17] "NCSU FreePDK 45nm Lib." https://eda.ncsu.edu/freepdk/freepdk45/.

[18] D. S. Richards, "Vlsi median filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 1, pp. 145–153, 1990.

[19] G. George, R. M. Oommen, S. Shelly, S. S. Philipose, and A. M. Varghese, "A survey on various median filtering techniques for removal of impulse noise from digital image," in *2018 Conference on Emerging Devices and Smart Systems (ICEDSS)*. IEEE, 2018, pp. 235–238.

[20] L. Yin, R. Yang, M. Gabbouj, and Y. Neuvo, "Weighted median filters: a tutorial," *IEEE Transactions on circuits and systems II: analog and digital signal processing*, vol. 43, no. 3, pp. 157–192, 1996.

[21] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–6.

ResISC: Residue Number System-Based Integrated Sensing and Computing for Efficient Edge AI

Sepehr Tabrizchi[†], Samin Sohrabi[†], Mohamadreza Mohammadi[‡], Ramtin Zand[‡], Shaahin Angizi[§], and Arman Roohi[†]
[†] Department of Electrical and Computer Engineering, University of Illinois Chicago, Chicago, IL, USA
[‡] Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA
[§] Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA
{stabr,ssohra2,aroohi}@uic.edu, mohammm@email.sc.edu, shaahin.angizi@njit.edu, ramtin@cse.sc.edu

***Abstract*—This paper presents ResISC, an RNS-based integrated sensing and computing architecture enabling efficient edge AI. ResISC platform features (i) an in-sensor residue encoder converting images directly to RNS in the analog domain, (ii) an energy-efficient RNS-based processing-near-sensor CNN accelerator utilizing SOT-MRAM, and (iii) an innovative mixed-radix unit for efficient activation operations. By employing selective channel deactivation, ResISC reduces computation overhead by up to 89%, while achieving a $3.4\times$ improvement in power efficiency and up to a $71\times$ reduction in execution time compared to processing-in-MRAM platforms. Experiments on various datasets demonstrate that ResISC achieves competitive accuracy levels (up to 94.63% on CIFAR-10) with minimal degradation, making it an ideal solution for power-constrained, real-time edge applications.**

I. Introduction

Imaging systems have become ubiquitous in today's technology landscape, embedded in mobile devices, safety systems, and various other applications, thanks to advancements in CMOS image sensors [1]. These sensors are the backbone of modern imaging technology, enabling high-quality photos and videos in compact devices. However, the continuous increase in the complexity and resolution of CMOS image sensors has resulted in larger data volumes and higher production costs [2]. Among the critical components of these sensors, Analog-to-Digital Converters (ADCs) play a vital role by converting continuous-time signals into discrete-time, binary-coded formats [3]. High-speed, high-resolution ADCs are costly and power-hungry, especially in portable devices [4], making it essential to minimize the number of ADC operations to decrease power dissipation and energy consumption. The proliferation of the Internet of Things (IoT) has led to a global network of over 75 billion devices projected to reach a market value of $1.1 trillion by 2025. Despite this growth, IoT devices often lack inherent intelligence and rely heavily on cloud-based decision-making, leaving a considerable portion of sensor-generated data unprocessed [5], [6]. This dependency raises concerns regarding data privacy, latency, and energy efficiency, especially in applications requiring real-time processing, such as autonomous vehicles and security systems [7]. Deploying complex machine learning models like Convolutional Neural Networks (CNNs) on edge devices poses substantial challenges due to their intensive storage and computational requirements. Edge devices typically have limited resources, making local implementation difficult. Consequently, many IoT devices depend on cloud-based processing, which introduces additional latency and energy consumption. This so-called CNN "power and memory wall" has motivated the exploration of alternative approaches to enhance CNN efficiency in both software and hardware domains.

In software strategies, techniques such as parameter quantization and network binarization [8] have been investigated to reduce computational complexity. Utilizing low bit-width weights and activations can significantly decrease the model size and computational demands [9]. Additionally, unconventional number systems like the Residue Number System (RNS) [10]–[13] have been explored to improve performance in computationally intensive applications. RNS offers unique features that accelerate computations by performing arithmetic operations on small residues independently and in parallel, reducing carry propagation and partial product sizes in multiplication. However, RNS is limited by the complexity of inter-modulo arithmetic operations and the need for interfacing with two's complement systems. From a hardware perspective, extensive research has been conducted on developing deep learning accelerators using GPUs, FPGAs, and ASICs. However, traditional architectures with isolated computing units and memory elements interconnected via buses face significant challenges, including limited memory bandwidth, long memory access latency, and high leakage power consumption [14]. Processing-in-Memory (PIM) architectures have been proposed to address these challenges by integrating logic computation within memory units, allowing data to be processed internally and exploiting large internal memory bandwidth [15]–[17]. To realize a thing-centered (data-centric) approach, wherein IoT nodes locally process data [18], recent efforts have focused on enhancing CMOS image sensors for faster processing of Deep Neural Network (DNN) workloads. Processing-Near-Sensor (PNS) and Processing-In-Sensor (PIS) paradigms integrate computation units with image sensors to process data before transmission [6], [19]–[21]. However, these approaches face limitations due to power-hungry peripherals, significant area overhead, and the energy consumption of ADCs, Digital-to-Analog Converters (DACs), and sense amplifiers, restricting the deployment of all DNN layers into the pixel array [5].

Our contributions are summarized as follows: (1) We embed a forward converter for RNS within the sensor, directly converting captured images into the RNS domain in analog form. This approach reduces the precision requirements of ADCs, thereby increasing system performance and efficiency by minimizing unnecessary data conversions and transfers. **(2)** We introduce an energy-efficient RNS-based processing-near-sensor CNN accelerator, termed *ResISC*. This accelerator

features fully RNS-based computational layers, leveraging the low-bit-width feature of RNS to reduce power consumption and latency. This design exploits large internal memory bandwidth and the parallelism inherent in Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM) to enhance speed and throughput. **(3)** To alleviate the overhead caused by inter-modulo operations in RNS, we develop a new efficient mixed-radix unit capable of performing activation operations. We achieve further improvements in the performance efficiency of systems, such as power and delay reduction, by optimizing image/modular channels read circuitry using a novel add-on. This design exploits large internal memory bandwidth and parallelism to enhance speed and throughput. **(4)** We evaluate system accuracy across different scenarios and assess system performance in terms of speed, throughput, and energy efficiency. Experimental results demonstrate that using **ResISC**, we achieve significant reductions in power consumption and delay with negligible accuracy degradation compared to existing platforms.

II. Backgrounds

A. Residue Number System (RNS)

A Residue Number System (RNS) is defined by a set of L pairwise coprime moduli $\{m_i\}$ for $i = 1, 2, \ldots, L$, where $L \geq 2$. The dynamic range M of the RNS is the product of all moduli, i.e., $M = \prod_{i=1}^{L} m_i$. Any unsigned integer $X \in [0, M)$ can be uniquely represented by an L-tuple $(x_1, x_2, \ldots, x_L)$, where each residue x_i is the least positive remainder when X is divided by m_i, denoted as $x_i = X \bmod m_i$. Signed integers $\hat{X} \in [-M/2, M/2)$ use the representation $\hat{X} = \langle X + \lfloor M/2 \rfloor \rangle_M - \lfloor M/2 \rfloor$ to relate X and its signed version within the same moduli-set. Let A, B, and R be integers represented by residue sequences $(a_1, a_2, \ldots, a_L)$, $(b_1, b_2, \ldots, b_L)$, and $(r_1, r_2, \ldots, r_L)$, respectively. R results from applying an arithmetic operation $\otimes \in \{+, -, \times\}$ to A and B. In the RNS domain, R is computed as $(r_1, r_2, \ldots, r_L) = (\langle a_1 \otimes b_1 \rangle_{m_1}, \langle a_2 \otimes b_2 \rangle_{m_2}, \ldots, \langle a_L \otimes b_L \rangle_{m_L})$, with $\langle a_i \otimes b_i \rangle_{m_i}$ indicating modulo operation $\otimes$ on residues a_i and b_i. RNS arithmetic operations occur in parallel across independent channels, offering efficiency over two's-complement arithmetic. However, certain operations, known as *inter-modulo operations*—such as reverse conversion, sign detection, magnitude comparison, and scaling—cannot be performed independently in each residue channel. These operations require algorithms that map the residue representation back to its equivalent weighted integer. One prominent method is the Mixed Radix Conversion (MRC), which reconstructs the integer X from its residues as [22]: $X = \sum_{i=1}^{L} \left(v_i \prod_{j=1}^{i-1} m_j \right)$, where v_i are the mixed-radix coefficients calculated sequentially using the recurrence relations:

$$F_{i,j} = \begin{cases} x_j, & i = 1, \\ (F_{i-1,j} - v_{i-1}) \langle m_{i-1}^{-1} \rangle_{m_j}, & i > 1, \end{cases} \tag{1}$$

with $v_i = \langle F_{i,i} \rangle_{m_i}$, and $\langle m_a^{-1} \rangle_{m_b}$ being the multiplicative inverse of m_a modulo m_b. For a 3-moduli RNS, the integer X can be represented using MRC as: $X = v_1 + m_1 v_2 + m_1 m_2 v_3$. The mixed-radix coefficients v_i are computed as, $v_1 = x_1$, $v_2 = \left\langle (x_2 - x_1) \langle m_1^{-1} \rangle_{m_2} \right\rangle_{m_2}$, $v_3 = \left\langle \left[(x_3 - x_1) \langle m_1^{-1} \rangle_{m_3} - v_2 \right] \langle m_2^{-1} \rangle_{m_3} \right\rangle_{m_3}$.

B. RNSed CNN Acceleration

Recent advancements in addressing the computational challenges of deep neural networks (DNNs) have highlighted the RNS as a promising computing paradigm for accelerating compute-intensive applications [23], [24]. Research from both academia and industry has demonstrated the advantages of RNS in this context. In [25], [26], a nested RNS approach was used to reduce bit-widths, resulting in efficient implementations of lookup tables (LUTs). However, this approach encountered limitations due to significant data movement between memory and computation units, alongside high memory requirements. An RNS-based artificial neural network was proposed in [27], where neurons were modeled using ROMs. Although this method improved computational efficiency, its scalability was limited by the number of input neurons and the memory storage required. In another study [28], hardware complexity for multiply-accumulate (MAC) operations in convolutional (CONV.) layers was reduced by leveraging RNS-based approximate computation. This approach implemented only the MAC operation in the RNS domain, requiring multiple conversions between the Binary Number System (BNS) and RNS to function correctly. Such conversions introduced performance overheads. The work in [29] implemented CONV layers in the RNS domain on FPGAs, with activation and pooling computations executed in the software domain. Similar to [28], [30], this method required N BNS-to-RNS and RNS-to-BNS conversions before and after each CONV layer (where N is determined by the number of CONV layers), leading to significant resource and delay overheads. In [31], the RNSnet architecture was proposed, employing a memristor-based DNN fully in the RNS domain. RNSnet mapped all parameters to RNS, performing MAC, activation, and pooling operations without requiring intermediate BNS/RNS conversions. While LUTs were utilized for partial multiplications, LUT-based implementations in RNSnet were limited in generalizability across different channel bit lengths and provided limited pipelining capability. Additionally, performing multiplications required three add operations per channel, leading to considerable delays. The Res-DNN architecture, introduced in [32], implemented all primary components of the convolutional layer in the RNS domain, drawing inspiration from the Eyeriss architecture and employing Huffman coding to minimize memory usage. A primary limitation of this design was its reliance on direct module-by-module mapping, necessitating specialized hardware units for inter-modulo operations such as sign detection and comparison to implement activation and MAX pooling layers. The need for dedicated components for each function imposed additional overhead, reducing the overall performance benefits of the RNS-based architecture.

III. Proposed ResISC architecture

ResISC is designed to enhance the advantages of RNS and mitigate its disadvantages as much as possible. To achieve

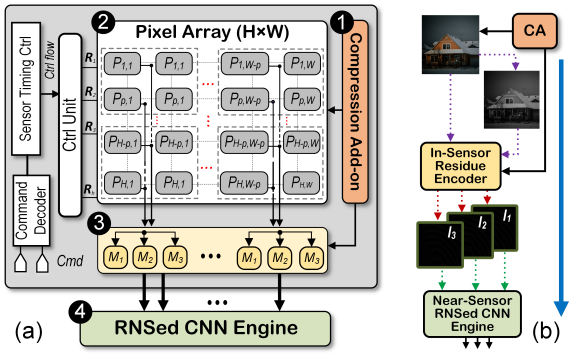


Fig. 1. (a) The proposed ResISC architecture and (b) its workflow.

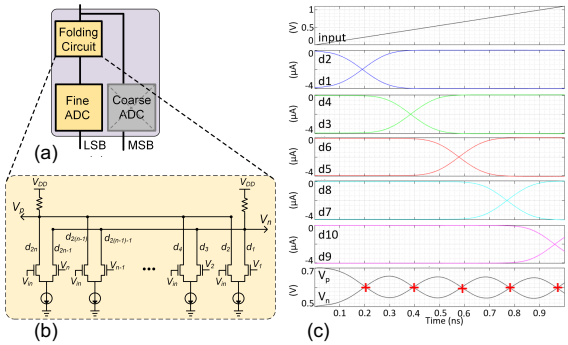


Fig. 2. (a) The Folding ADC consists of a folding circuit and a fine ADC. (b) Circuit level of the folding building block. (c) Transient vector for a folding circuit with 5 arms.

this, we minimize RNS overhead wherever feasible. In this design, forward conversion is eliminated by using the proposed RNSed-ADC, while our RNSed CNN engine handles reverse conversion. There is an innovative optimizer, called the Compression Add-on (CA) to enhance the overall system's efficacy. Additionally, ResISC utilizes a specific moduli set to enable some non-RNS operations, like Sign, within the RNS domain. The architecture of ResISC is depicted in Fig. 1, where, in ❶, the CA selects the required color and moduli channels. Next, in ❷, an image is captured by the image sensor. The image is converted within the sensor to RNS values on a row-by-row basis in ❸. Finally, in ❹, these RNS values are applied to the RNS engine as inputs to perform near-sensor computation/PIM. Each of these components is explained in detail in the following sections.

A. Compression Add-on

This component is responsible for managing the other components. The compression level can be divided into two distinct levels. The first level focuses on the pixel array colors/photodiodes, including red, green, and blue. The CA can deactivate one or two colors in the pixel array, resulting in an image composed of only one or two colors. Studies have shown that, in some applications, only a single color channel is sufficient for image recognition and classification [33]. In this case, all the related circuits, including ADCs and RNSed engine channels, are turned off, leading to a significant reduction in power. The second level of optimization is applied only when the first level has been satisfied or successfully implemented. At this level, only one of the colors is enabled and in addition, due to the parallelism inherent in each RNS moduli channel, the CA can independently deactivate each moduli channel. Consequently, when a moduli channel is deactivated, the corresponding moduli channel on the RNSed-ADC and corresponding CNN sub-network on the RNSed engine are also turned off. In such cases, not only the corresponding ADCs and sub-networks are deactivated, but the relevant pixels are also turned off. This approach can reduce computation by up to 89%. It should be mentioned this optimization has the same effect on on-chip communication.

B. In-Sensor Residue Encoder

There are several types of ADCs, including flash, successive-approximation register (SAR), and folding ADCs, each with distinct characteristics. Flash ADCs offer high speed but are power-intensive, whereas SAR-ADCs are energy-efficient but relatively slow. In this work, we present the RNSed-ADC, a folding ADC that we have specifically modified for the RNS domain. Folding-ADCs partition the n-bit input into two parts: coarse and fine. The coarse part handles the Most Significant Bits (MSB) of the input, while the fine circuit generates the Least Significant Bits (LSB). The complete architecture of a traditional Folding-ADC is illustrated in Fig. 2(a).

1) Folding: In the folding ADC, the fine section operates independently, allowing for the coarse part to be omitted when working in the RNS domain. For example, considering a modulus of 5, the output sequence cycles through $\{0, 1, \ldots, 4, 0, 1, \ldots\}$ as the input increases. This cyclical behavior aligns with the folding circuit's capability to mimic such patterns. By incrementally increasing the input, the circuit produces a sine-wave-like output within a specific range, replicating the desired behavior for RNS applications. A key characteristic of RNS is that upon reaching a certain modulus, the output cycles through a predefined sequence before repeating. The architecture of an RNS-based Folding ADC is depicted in Fig. 2(b). The number of pair transistors in this architecture is presented as the number of repetitions in the wave, which is equal to the integer divisor. For example, for an 8-bit value (up to 256) in moduli of 8, we need to have 32 arms, each comprising two transistors and a current source. The current source in each arm ensures that the current remains constant, which is crucial for maintaining stable operation throughout the system. Additionally, each arm is associated with a distinct reference voltage, allowing for precise control over the modulation process. The relationships among the linear input signal, the current through each arm, and the output signals for five arms are further illustrated in Fig. **??**. This configuration highlights the intricate interplay between the input signal, current distribution among the arms, and the resulting RNS-modulated output.

2) Encoder: In Fig. 2(c), at the points where the positive (V_p) and negative (V_n) outputs intersect, the modulus resets to zero. To divide the regions between these crossover points into equal segments according to the modulus, folding circuits are utilized with adjusted reference voltages to achieve the required spacing. For instance, with a modulus of 5, five folding circuits with shifting reference voltage are added to ensure equal division. To optimize the power efficiency of the RNSed-

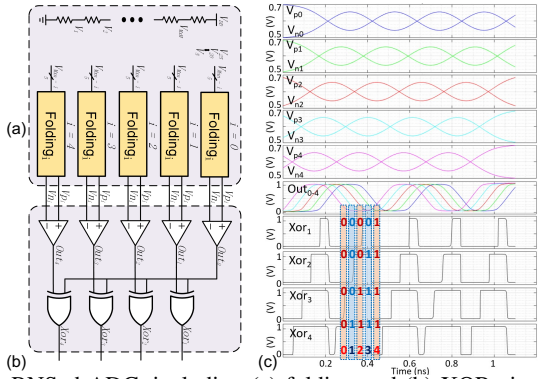


Fig. 3. RNSed-ADC, including (a) folding and (b) XOR circuits. (c) Results for five folding circuits and the XOR array.

ADC, a single shared resistor ladder is employed across all folding circuits. The structural layout, interconnections, and resultant outputs of this configuration are depicted in Fig. 3, providing a visual representation of the system's operational framework. As shown in Fig. 3, each pair of positive and negative outputs is compared using a comparator. The results of these comparisons are collectively displayed in Fig. 3(c) as Out_{0-4}. Following this, an XOR operation is performed on these outputs, as depicted in Fig. 3(b). Specifically, the outputs Out_{1-4} are XORed with Out_0, resulting in the final RNSed-ADC output, denoted as Xor_{1-4} in Fig. 3(c). The presence of V_{DD} across these signals indicates the residue number; for example, if Xor_{1-4} yields the sequence "1110," the corresponding residue number is interpreted as 3.

C. Near-Sensor RNSed CNN Engine

Figure 4 shows the RNSed CNN Engine. The RNSed-ADC (1) converts the captured image to RNS, which is processed by the RNSed Convolution Engine (2), and the layer outputs are passed to the RNSed Inter-modulo Engines (3).

1) RNSed Convolution Engine (RCE): The RNS-Based CNN accelerator leverages the RNS to efficiently map CNN layers to hardware by reducing computational complexity and enabling high parallelism. Our engine operates on a three-moduli set $\{2^n - 1,\ 2^n,\ 2^{n+1} - 1\}$, chosen for optimal dynamic range and efficient inter-modulo operations. In this work, $n = 3$ is selected, resulting in moduli $\{7, 8, 15\}$. The CNN's weights and input feature maps (ifmap) are RNS-based values which mapped into computational sub-arrays for multiplications and accumulations processed in parallel channels within the RNS domain, where each channel corresponds to a specific modulus, convolution operations are executed based on PIM sub-arrays, termed the RNSed Convolution Engine (RCE). For PIM, we used SCiMA [16], which uses SOT-MRAM and reconfigurable sense amplifiers (SA) to support one-cycle addition and efficient multiplication. The RCE uses this architecture for convolution through optimized memory sub-arrays, enabling fully parallel processing of residues in each channel. Each memory bank has multiple computational memory sub-arrays linked to two shared modules, i.e., a global row decoder (GRD) and a global row buffer (GRB). Each sub-array is divided into data rows for storing operands and computation rows for performing operations using a modified row decoder, enabling multi-row activation to support in-memory operations. Results from the operations are stored in local row buffers, which are shared across sub-arrays. Control signals manage the sequence of operations, allowing the RCE to conduct RNS-based MAC computations. The RCE also includes a tailored instruction set that interprets commands from the memory controller, setting control signals for specific in-memory tasks and enabling efficient mapping of convolution data across sub-arrays. The final output of the CONV. layer is then transferred to the RNSed Inter-modulo Engine (RIE). With a specific instruction set for RNS operations, RNS achieves high-speed, energy-efficient CNN execution, significantly reducing energy consumption. This RNS-based approach demonstrates that RNS can deliver substantial power and performance benefits, making it highly suitable for power-constrained, real-time applications.

2) RNSed Inter-modulo Engine (RIE): RNSed Inter-modulo Engine (RIE) in the RNS accelerator addresses the complexities of performing activation and normalization within the RNS domain, which cannot be executed independently across parallel channels. Since the computation of inter-modulo is conducted using moduli and residues of more than one channel, a large area, and power-delay-product overheard are imposed on the RNS-based network. To mitigate this issue, we introduce an efficient approach to enhance hardware utilization of the proposed accelerator by producing mixed-radix coefficients by Mixed-Radix Engine (MRE). The general scheme of RIE comprises a Mixed-Radix Engine (MRE), an activation unit, and a scaling unit, which is shown in Fig. 4. A critical component of the RIE is the Mixed-Radix Engine (MRE), which generates mixed-radix coefficients used to simplify inter-modulo operations. These coefficients enable efficient implementation of functions like activation and normalization, significantly reducing resource demands for RNS-based accelerators. The MRE transforms each residue into mixed-radix coefficients (v_1, v_2, v_3) by leveraging specific modular arithmetic rules to determine the sign and scale of a weighted number, R. For activation, the finding in [34] the MSB of the coefficient v_3 indicates the Sign of R: if positive, the original values are retained; otherwise, they are set to zero—implementing the ReLU function, In this step, MSB(v_3) determines the proper output $\{0, 0, 0\}$ or $\{v_1, v_2, v_3\}$ negative or positive values respectively, and send it back to the MRE for the rest of RIE operations. If it passes $\{0, 0, 0\}$, other operations are skipped, which leads the performance improvement. The Scaler unit then normalizes values by adjusting magnitudes based on a targeted scaling factor, preventing overflow in subsequent layers. This scaling factor, often chosen as $2^{n+1}-1$, helps maintain efficient operations within the RNS domain. Together, the MRE and Scaler facilitate activation and normalization, optimizing computational resources and enabling effective hardware sharing across RNS channels.

IV. Evaluation Results

To assess the functionality and performance of ResISC, we adopt the moduli set $\{2^3 - 1, 2^3, 2^4 - 1\}$, yielding a dynamic

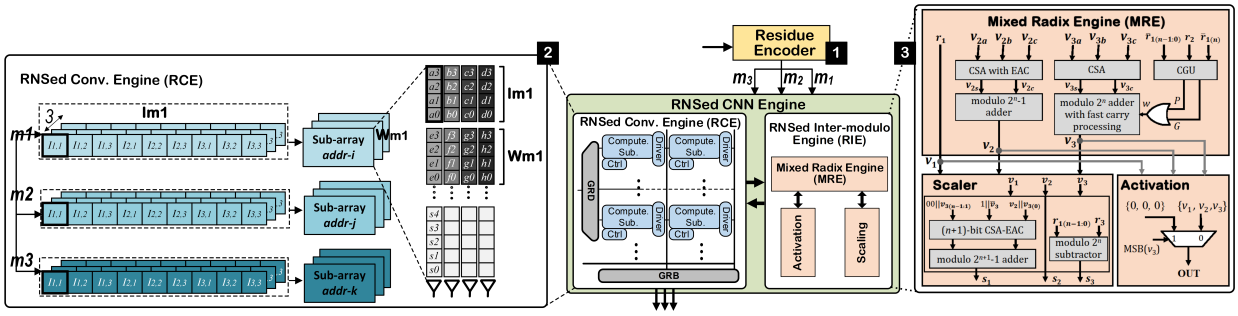


Fig. 4. Comprehensive overview of RNSed CNN engine, including RNSed conv. engine (RCE) and RNSed inter-modulo engine (RIE).

range of $[0, 840)$. This range is comparable to that of a 9-bit binary representation, which spans $[0, 512)$.

A. Framework

To assess the performance of the proposed accelerator, we developed an evaluation testbed from the ground up. At the circuit level, we implemented the ResISC's memory sub-array, along with associated peripheral circuits (e.g., sense amplifiers and a modified row decoder). This setup enables us to obtain the performance characteristics required for the multi-row activation approach. Additionally, we utilized Synopsys Design Compiler with 45nm technology to determine the energy and area parameters. Among all components, the MRE demands the most power and area due to its complexity, while the activation unit, which includes a simple 2-to-1 multiplexer (MUX), exhibits the lowest power and area requirements. In this network, we use 4 memory banks, each of which is composed of 4×4 matrices (MATs). Each MAT contains several computational memory sub-arrays interconnected through a shared GRB and a GRD. This memory is configured in 1024 rows and 256 columns, which means this memory can do 2^{14} operations in one cycle.

B. Performance Evaluation

In ResISC, we employ three key techniques to enhance performance: RNS-ADC, PIM, and the CA component, as illustrated in Fig. 5. First, using three smaller RNSed-ADCs results in significant power and delay savings compared to a larger ADC. To validate this, we simulated an 8-bit flash ADC and three- and four-bit RNSed-ADCs in HSPICE using the 45nm PTM library. The normalized delay and power consumption results are depicted in Fig. 5(a). It should be noted that the total power consumption of the RNSed-ADCs is cumulative, while their delay corresponds to that of the four-bit RNSed-ADC. As shown in the figures, the delay and power consumption are improved by approximately 50% and 19%, respectively. Figure 5(b) reports and compares the execution time of ResISC with the under-test processing-in-MRAM platforms, i.e., SCiMA [16], IMCE [15], Pinatubo [35], and MPIM [36]. We observe that the ResISC platform processes all the convolutional, Residual, and fully-connected layers in less than $8ms$. It is noteworthy that as we are using three parallel channels corresponding to $\{m_1 = 7, m_2 = 8, m_3 = 15\}$, the network execution time can be well estimated by the longest channel, i.e., m_3. In this way, ResISC reduces the execution time by a factor of ∼6.4×, 30×, 71×, and 56.6× compared with SCiMA, IMCE, Pintatubo, and MPIM implementations. This advantage is achieved because (1) the proposed in-memory architecture is able to process the main components of adder trees based on RCE very efficiently in 1 or 2 cycles in parallel compared with multi-cycle operations in processing-in-MRAM counterparts and (2) the parallel mapping strategy matches the behavior of adder trees. We report the estimated power consumption of various PIM platforms for running the same network. The power is reported by adding on the power consumed in three parallel channels. Based on Fig. 5(c), ResISC with the presented MRAM implementation in this work shows the least power consumption (total over three channels) to run the ResNet-18, as compared with the MRAM-PIMs. The ResISC reduces the power consumption by ∼3.4× compared with the SCiMA implementation. In this evaluation setup, the quantization level is adjusted to 10-bit. Consequently, the operations are reduced compared to a 32-bit implementation on SCiMA. On the other hand, IMCE, Pinatubo, and MPIM are not able to perform the addition in one cycle. As a result, we need to use other basic operations like NAND to calculate the addition. This weakness increases the number of writing the temporary values in memory, which consumes much more power. It is worth pointing out that the analysis reveals that the first five layers account for the majority of the delay and power consumption, rendering the contributions of the remaining layers negligible. The last bar, shown in Fig. 5 (d), presents the normalized energy savings utilizing the CA component. This figure compares the system's power consumption in full-color mode (RGB) with various scenarios. As illustrated, disabling specific color channels reduces the system's power consumption. Regarding power consumption, there is no significant difference between the color channels. However, a difference emerges when removing a single modulus. On the other hand, modules have distinct bit lengths, represented by separate bars. As expected, disabling the modulus of 16 achieves greater power savings than the other modules because of the bigger length.

C. Accuracy and Robustness

To evaluate the performance of ResISC, we implemented all components in Python and conducted experiments using ResNet-18 on CIFAR-10, SVHN, and CIFAR-100, as summarized in Table I. In this table, the baseline corresponds to the RGB column, where the network is trained using

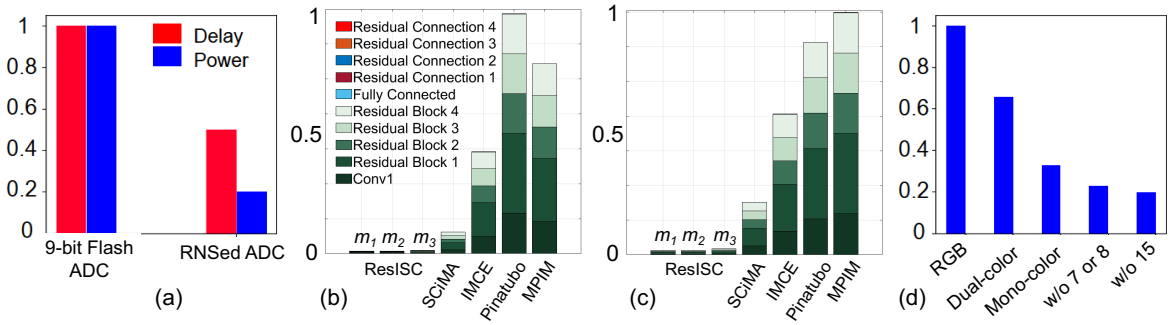


Fig. 5. Performance evaluation, including ADC (a) delay and power consumption. The breakdown of (b) execution time (S) and (c) power (W) of ResISC compared with four PIM accelerators. (d) The power consumption of different ResISC implementations regarding the CA.

TABLE I
ACCURACY OF RESISC ON SVHN, CIFAR-10 AND CIFAR-100 DATASETS.

	RGB	RG	RB	GB	R	G	B	R-7	R-8	R-15	G-7	G-8	G-15	B-7	B-8	B-15
SVHM	96.31	96.5	96.38	96.41	96.16	96.45	95.99	93.35	93.53	93.39	93.81	93.4	93.45	92.89	93	93.02
CIFAR10	94.63	94.89	93.41	94.76	93.41	93.62	93.04	93.22	93.5	93.53	93.57	93.34	93.61	93.18	92.95	93.08
CIFAR100	75.87	74.22	74.34	72.29	71.64	71.25	70.19	71.95	71.95	71.68	71.57	71.53	71.63	69.94	69.69	70.11

TABLE II
INFERENCE ACCURACY (%) ACROSS DIFFERENT DESIGNS.

Configuration	MNIST	SVHN	CIFAR-10
32-bit FP	99.46	97.52	95
RNSiM [37]	99.05	96.2	91.7
Residue-Net [38]	99.3	–	93.59
BNN	98.60	91.1	89.8
ResISC	99.39	96.31	94.63

the original images without any modification. Utilizing the proposed CA method, ResISC optimizes the pixel array by selectively activating one or two colors in the first step. This approach results in three dual-color combinations (RG, RB, GB) and three single-color combinations (R, G, B). As shown in Table I, for SVHN and CIFAR-10, certain dual-color combinations achieve even higher accuracy compared to the baseline by removing unnecessary color information. For example, in CIFAR-10, the Red and Green (RG) combination achieves an accuracy of 94.89%, which is a 0.26% improvement over the baseline. Similarly, in the SVHN dataset, the mono-color Green (G) achieves a 0.24% accuracy increase. However, for larger datasets such as CIFAR-100, there is a slight accuracy drop in dual-color (approximately 1%). Despite this, the computation cost is reduced by up to 66%, making this method ideal for low-power applications. The second level of optimization is detailed in the rest of the table. Here, the first letter (R, G, B) indicates the active color, and the subsequent number represents the modulus that is turned off. For instance, in R-7, only the red channel remains active while modulus 7 is deactivated. Since RNS enables each channel to operate independently, results from individual channels can be combined. In this case, we consider the lost channel as missing data and initially assign it a random value within the modulus range. This approach yielded a maximum accuracy of 73% on CIFAR-10, which was deemed unacceptable. To improve performance, we replaced the random value with a constant value of zero, leading to significant improvements. As shown in Table I, the accuracy drop for CIFAR-10 and CIFAR-100 is less than 1% and 4%, respectively, while computation is reduced by approximately 40%. It should be mentioned that all the subnetworks operate in the RNS domain. Based on this, turning off the moduli of 15 achieves more improvement in the case of power consumption than other modules. This model needs 4 bits for computation, while others need only 3 bits. As shown in Table II, the inference accuracy of various configurations is compared on the MNIST, SVHN, and CIFAR-10 datasets. For the 32-bit FP configuration, accuracies of 99.46%, 97.52%, and 95% are achieved on MNIST, SVHN, and CIFAR-10, respectively. Our proposed ResISC delivers competitive results, achieving 99.39%, 96.31%, and 94.63% on the same datasets, demonstrating its ability to maintain high performance while reducing computational complexity. In comparison, other RNS-based CNN accelerators, such as RNSiM, achieve 99.05% on MNIST, 96.2% on SVHN, and 91.7% on CIFAR-10. Residue-Net attains 99.3% on MNIST and 93.59% on CIFAR-10. Binary Neural Networks (BNN) exhibit a trade-off in accuracy, performing lower than others, with its highest accuracy being 98.6% on MNIST. Notably, MNIST is excluded from ResNet18-specific evaluations due to its simplicity and the ease of achieving over 99% accuracy.

V. CONCLUSION

ResISC demonstrates an efficient RNS-based integrated sensing and computing architecture for edge AI applications. Our mixed-radix unit and in-sensor residue encoder enable efficient CNN operations with minimal accuracy degradation. Experimental results confirm up to 89% computation reduction through channel deactivation and a 3.4× power efficiency improvement compared to state-of-the-art platforms. Moreover, ResISC maintains high accuracy across diverse datasets, with a less than 1% accuracy drop while reducing computation by up to 66%. The ResISC architecture presents a groundbreaking RNS-based framework for edge AI, capable of being extended to various tasks. It enhances data privacy by reducing cloud dependency and bandwidth requirements.

ACKNOWLEDGMENT

This work is supported in part by SRC and the National Science Foundation (NSF) under grant numbers 2216772, 2448133, 2504839, 2340249, and 2228028.

REFERENCES

[1] S. K. Mendis, S. E. Kemeny, R. C. Gee, B. Pain, C. O. Staller, Q. Kim, and E. R. Fossum, "Cmos active pixel image sensors for highly integrated imaging systems," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 2, pp. 187–197, 1997.

[2] J. Choi *et al.*, "An energy/illumination-adaptive cmos image sensor with reconfigurable modes of operations," *IEEE JSCC*, vol. 50, no. 6, pp. 1438–1450, 2015.

[3] E. R. Fossum, "Cmos image sensors: Electronic camera-on-a-chip," *IEEE transactions on electron devices*, vol. 44, no. 10, pp. 1689–1698, 1997.

[4] J. Liu, Z. Luo, and X. Xiong, "Low-resolution adcs for wireless communication: A comprehensive survey," *IEEE Access*, vol. 7, pp. 91 291–91 324, 2019.

[5] R. Song *et al.*, "A reconfigurable convolution-in-pixel cmos image sensor architecture," *IEEE TCSVT*, 2022.

[6] H. Xu *et al.*, "Macsen: A processing-in-sensor architecture integrating mac operations into image sensor for ultra-low-power bnn-based intelligent visual perception," *IEEE TCAS II*, vol. 68, pp. 627–631, 2020.

[7] R. Hussain and S. Zeadally, "Autonomous cars: Research results, issues, and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1275–1313, 2018.

[8] S. Zhou *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[9] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*. Springer, 2016, pp. 525–542.

[10] C.-H. Chang *et al.*, "Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications," *IEEE circuits and systems magazine*, vol. 15, pp. 26–44, 2015.

[11] N. Taheri, S. Tabrizchi, S. Angizi, and A. Roohi, "Chaosen: Security enhancement of image sensor through in-sensor chaotic computing," in *2024 IEEE 42nd International Conference on Computer Design (ICCD)*. IEEE, 2024, pp. 1–8.

[12] S. Tabrizchi, N. Taheri, S. Angizi, and A. Roohi, "Racsen: Residue arithmetic and chaotic processing in sensors to enhance cmos imager security," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 551–555.

[13] N. Taheri, S. Tabrizchi, D. Najafi, S. Angizi, and A. Roohi, "Ressen: Imager privacy enhancement through residue arithmetic processing in sensors," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2024, pp. 349–354.

[14] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[15] S. Angizi, Z. He, F. Parveen, and D. Fan, "Imce: Energy-efficient bitwise in-memory convolution engine for deep neural network," in *Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific*. IEEE, 2018, pp. 111–116.

[16] S. Tabrizchi, S. Angizi, and A. Roohi, "Scima: A generic single-cycle compute-in-memory acceleration scheme for matrix computations," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 576–580.

[17] S. Tabrizchi, M. Morsali, D. Pan, S. Angizi, and A. Roohi, "Pinsim: A processing in-and near-sensor simulator to model intelligent vision sensors," *IEEE Computer Architecture Letters*, 2024.

[18] T.-H. Hsu *et al.*, "AI edge devices using computing-in-memory and processing-in-sensor: from system to device," in *IEDM*, 2019.

[19] S. J. Carey *et al.*, "A 100,000 fps vision sensor with embedded 535gops/w 256× 256 simd processor array," in *Symposium on VLSI*. IEEE, 2013.

[20] T.-H. Hsu, Y.-R. Chen *et al.*, "A 0.5-v real-time computational cmos image sensor with programmable kernel for feature extraction," *IEEE JSSC*, vol. 56, pp. 1588–1596, 2020.

[21] S. Angizi *et al.*, "Pisa: A non-volatile processing-in-sensor accelerator for imaging systems," *IEEE TETC*, 2023.

[22] M. Taheri *et al.*, "Efficient incorporation of the rns datapath in reverse converter," *IEEE TCAS II: Express Briefs*, 2020.

[23] M. Abdelhamid and S. Koppula, "Applying the residue number system to network inference," *arXiv preprint arXiv:1712.04614*, 2017.

[24] Z.-G. Liu and M. Mattina, "Efficient residue number system based winograd convolution," *arXiv preprint arXiv:2007.12216*, 2020.

[25] H. Nakahara *et al.*, "A high-speed low-power deep neural network on an fpga based on the nested rns: Applied to an object detector," in *ISCAS*. IEEE, 2018.

[26] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2015, pp. 1–6.

[27] G. Alia and E. Martinelli, "Neurom: a rom based rns digital neuron," *Neural networks*, vol. 18, no. 2, pp. 179–189, 2005.

[28] V. Arrigoni *et al.*, "Approximate operations in convolutional neural networks with rns data representation." in *ESANN*, 2017.

[29] N. I. Chervyakov *et al.*, "Area-efficient fpga implementation of minimalistic convolutional neural network using residue number system," in *2018 FRUCT*. IEEE, 2018, pp. 112–118.

[30] N. Chervyakov *et al.*, "Residue number system-based solution for reducing the hardware cost of a convolutional neural network," *Neurocomputing*, 2020.

[31] S. Salamat *et al.*, "Rnsnet: In-memory neural network acceleration using residue number system," in *2018 ICRC*. IEEE, 2018, pp. 1–12.

[32] N. Samimi *et al.*, "Res-dnn: A residue number system-based dnn accelerator unit," *IEEE TCAS I: Regular Papers*, vol. 67, pp. 658–671, 2019.

[33] S. Tabrizchi *et al.*, "Appcip: Energy-efficient approximate convolution-in-pixel scheme for neural network acceleration," *IEEE JETCAS*, vol. 13, pp. 225–236, 2023.

[34] C. V. Niras and Y. Kong, "Fast sign-detection algorithm for residue number system moduli set {2n- 1, 2n, 2n+ 1- 1}," *IET Computers & Digital Techniques*, vol. 10, no. 2, pp. 54–58, 2016.

[35] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[36] M. Imani, Y. Kim, and T. Rosing, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 757–763.

[37] A. Roohi, M. Taheri, S. Angizi, and D. Fan, "Rnsim: Efficient deep neural network accelerator using residue number systems," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[38] S. Salamat, S. Shubhi, B. Khaleghi, and T. Rosing, "Residue-net: Multiplication-free neural network by in-situ no-loss migration to residue number systems," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 222–228.

Enhancing Parallelism and Energy-Efficiency in SOT-MRAM based CIM Architecture for On-Chip Learning

Anubha Sehgal*, Alok Kumar Shukla†, Sumit Diware‡, Sandeep Soni*, Seema Dhull§, Sonal Shreya¶, Sourajeet Roy*, Rajendra Bishnoi‡
* Indian Institute of Technology Roorkee, † Madan Mohan Malaviya University of Technology
‡ Delft University of Technology, § Global Foundaries, ¶ Aarhus University
{anubha_s, sandeep_s, sourajeet.roy}@ece.iitr.ac.in, aks.ece@mmmut.ac.in, seema@globalfoundaries.com,
{s.s.diware, r.k.bishnoi}@tudelft.nl, sshreya@ece.au.dk

***Abstract*—Computational-In-Memory (CIM) architectures have emerged as energy-efficient solutions for Artificial Intelligence (AI) applications, enabling data processing within memory arrays and reducing the bottleneck associated with data transfer. The rapid advancement of AI demands real-time on-chip learning but implementing this with CIM architectures poses significant challenges, such as limited parallelism and energy-efficiency during training and inference. In this paper, we propose a novel CIM architecture specifically designed for on-chip learning applications, which capitalizes on the unique properties of Spin-Orbit Torque (SOT) technology to enhance both parallelism and energy-efficiency in computation. The proposed architecture incorporates a bulk-write mechanism for SOT-cell based arrays, enabling efficient weight updates during on-chip training. Additionally, we develop a scheme to process vector elements concurrently for vector-matrix multiplications during inference. To achieve this, we design multi-port bit-cell access capabilities along with their associated control mechanisms. Simulation results show a 5.82× reduction in latency and a 3.20× improvement in energy-efficiency compared to standard SOT-MRAM based CIM, with negligible overhead.**

***Index Terms*—Spin-Orbit Torque, multi-port memory, on-chip learning, neural networks, computation-in-memory**

I. Introduction

The rising demand for large AI datasets, along with advancements in edge computing and Internet-of-Things devices, has amplified the need for energy-efficient hardware architectures. Traditional computing systems rely heavily on data movement between processing and memory units, facing memory-wall challenges like high energy consumption and latency [1], [2]. To address these challenges, *Computing-In-Memory* (CIM) presents a promising solution by integrating memory and computation within the same hardware unit that minimizes the data movement [3]–[7]. Spintronic devices, especially *Spin-Orbit Torque Magnetic Random Access Memory* (SOT-MRAM), are ideal for such architectures due to their high-density data storage and energy-efficient operation [8]–[11]. SOT-MRAM based CIM architectures use the resistance states of SOT devices for analog matrix-vector multiplications within memory, enhancing efficiency and reducing access latency during their write operations [12]. Although such CIM architectures offer good parallelism, the demand for high computational throughput is much greater, especially for implementing on-chip learning, which requires continuous real-time updates directly within the hardware.

Limited work has been done to enhance the parallelism of CIM architectures for on-chip learning. For instance, the work proposed in [13] facilitates N parallel MAC operations across columns in the memory array by clamping the source lines with low-power adjustable resolution *Analog to Digital Converters* (ADCs) for enhancing parallelism in *Resistive Random Access Memory* (ReRAM) based CIM architecture. A higher ADC resolution leads to better system recognition accuracy while consuming more energy and latency. Moreover, a voltage-based differential referencing-in-array scheme has been proposed to enable multi-operand logic operations with high parallelism. However, this technique is limited to logic operations and does not apply to *Multiply-and-Accumulate* (MAC) operations [14]. Several approaches aim to boost the energy-efficiency of CIM architectures for on-chip learning. For example, an energy-efficient heterogeneous multicore processor proposed in [15] is designed for a deep belief network, equipped with transposable weight memory for on-chip local learning. Nevertheless, it encounters bottlenecks, including the area-inefficient transposable memory based on registers and limited parallelism caused by core utilization degradation. Moreover, an *Static Random Access Memory* (SRAM) based CIM architecture proposed in [16] uses pipeline optimization for CIM training to improve the energy-efficiency. However, the periphery circuit of transposable systems can also introduce complexity and latency during online training, limiting their effectiveness in real-time applications. Since most existing methods either demand high energy or suffer from slow read/write operations, there is a need for a low-latency, energy-efficient CIM solution that ensures high parallelism and supports efficient on-chip learning.

In this paper, we present a novel SOT-MRAM based CIM architecture designed for on-chip learning that leverages the

This work was supported by the Scheme for Promotion of Academic and Research Collaboration (SPARC), Grant No: SPA-2136-ECD-CNA/23-24.

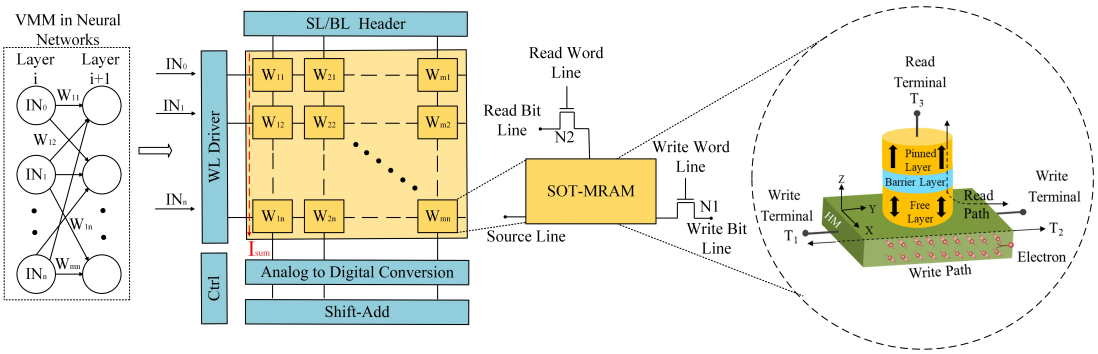


Fig. 1. Vector-matrix multiplication in Computation-In-Memory (CIM) architecture using SOT-MRAM bit-cell for neural networks.

unique features of the bit-cell to enhance parallelism and energy-efficient computation. Our contributions for this paper are as follows:

- We propose a bulk-write mechanism that facilitates effective weight updates during on-chip training for SOT-MRAM bit-cell based arrays.
- We introduce a concurrent processing scheme for vector-matrix multiplication, utilizing a multi-port SOT-MRAM bit-cell and their associated control mechanisms during inference.
- We perform a comprehensive evaluation of our system architecture for CIM-based image classification.

Simulation results demonstrate that our proposed approach can reduce latency by 5.82× and improve energy-efficiency by 3.20× compared to standard single-port SOT-MRAM based CIM architecture.

The rest of the paper is organized as follows: Section II presents the background of CIM architecture, SOT-MRAM technology, and on-chip learning. Section III presents the proposed CIM architecture, and Section IV illustrates the results. Finally, Section V concludes the paper.

II. Background

A. Computation-In-Memory Architecture

CIM architecture addresses data transfer bottlenecks by performing the operations and storage in the same physical location, thereby overcoming the overhead associated with costly data movements inherent in conventional von Neumann architectures [17]–[19]. Fig. 1 demonstrates the CIM architecture for neural networks employing *Vector-Matrix Multiplication* (VMM). The network weights are mapped as resistance states of bit-cells in the crossbar array. Multiple inputs are applied simultaneously to the bit-cells in each row, which are then multiplied by weights and generate current. The resulting current values are summed up in a column to execute VMM operations and then converted to digital outputs using ADCs. This approach provides high computational parallelism and improves energy-efficiency [20].

B. Spin-Orbit Torque Technology

A SOT-MRAM cell stores data in the form of magnetic resistance having a low switching current and faster access speed [21], [22]. It consists of a barrier oxide layer sandwiched between two ferromagnetic layers, a free and pinned/fixed layer, as shown in Fig. 1. The magnetic orientation of the pinned layer is always fixed, while that of the free layer can be freely rotated depending on the direction of the current. It is a three-terminal device having isolated read and write terminals that eliminate the oxide barrier breakdown issue. The write current is applied between two write terminals (T_1 and T_2) through heavy metal that generates spin-orbit torque on the magnetization of the free layer. This torque causes the magnetic orientation of the free layer to change, resulting in a change in resistance values due to the *Spin Hall Effect* (SHE) and Rashba effect [23]–[25]. When the magnetic orientations of the free and pinned layers are aligned in the same direction, the MRAM cell exhibits a low resistance state while the antiparallel exhibits a high resistance state. For read operations, a low read current is applied between the read terminal (T3) and the write terminal (T2). This read current is sensed by a sense amplifier to determine the stored value. A typical bit-cell structure is shown in Fig. 1.

C. On-Chip Learning

On-chip learning enables a computing system to execute machine learning tasks directly within the hardware where the data is processed. Integrating learning algorithms with memory elements such as SOT-MRAM devices on the same chip facilitates real-time updates and adaptations. The training process involves four steps, namely 1) feed-forward, 2) backpropagation for error calculation, 3) weight gradient calculation, and 4) weight update [26], [27]. During forward propagation, data is processed and stored in memory cells, which perform computations like matrix multiplications. The intermediate activations in each layer are stored in the buffers and the error between the predicted output and the label is calculated. During backpropagation, the weight gradients of each layer are calculated based on learning algorithms by convolution between the error and the activation obtained in the feed-forward process. Finally, the weights of the current layer are updated by the learning rate. For inference, the system uses the trained model to make predictions directly, thus reducing latency and energy consumption by minimizing

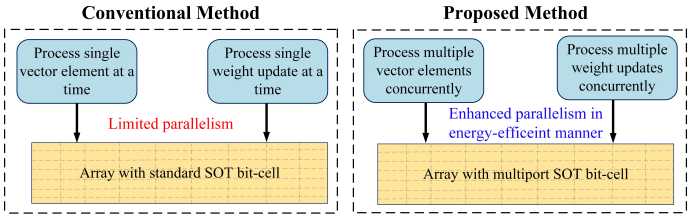


Fig. 2. Overview of conventional and proposed SOT-MRAM architectures.

data transfer. This integration enhances efficiency and enables adaptive, real-time learning and decision-making [28].

III. Proposed SOT-MRAM based CIM Architecture

In this section, we first present an overview of our proposed CIM architecture for on-chip learning with a detailed explanation of bit-cell design, control circuit, and weight calculation block. Subsequently, we provide the design details for weight update and inference.

A. Overview

As discussed, the on-chip training process requires real-time weight updates. For a SOT-MRAM based CIM architecture, this process is primarily divided into two components, namely, weight updates and the inference engine. As shown in Fig. 2, in conventional methods, weight updates are performed by programming cells one at a time, means the number of cycles required is equal to the number of write operations. Additionally, during inference, conventional approaches process one vector element at a time for vector-matrix multiplication, as shown in Fig. 2. This limits parallelism in conventional SOT-MRAM based CIM architectures, making them not suitable for fast real-time applications such as real-time ECG monitoring, autonomous vehicle control, and drone navigation, all of which demand rapid processing. In the proposed method, we utilize a bulk-write technique to write multiple cells simultaneously. Due to the symmetric resistance characteristics of write-1 and write-0 operations in SOT-MRAM cell, we can drive higher currents (depending on the technology and architecture), allowing for more efficient programming. To enhance the inference engine, we process multiple vector elements concurrently. This is achieved by modifying the existing SOT-MRAM bit-cell design with a multi-port bit-cell configuration. As a result, the proposed architecture offers greater parallelism and improved performance for on-chip learning applications. The next subsections detail the architecture and its components.

B. Architecture Design

Fig. 3 presents the macro-level architecture, illustrating the composition of a single CIM tile. Each tile executes the MAC operation that involves the input and the weights stored in the SOT-MRAM bit-cell. Multiple tiles can be cascaded together to form a fully connected neural network. Each tile consists of mainly the SOT-MRAM bit-cell, the control circuit, data converters, and weight calculation/update blocks. The core of the architecture utilizes SOT-MRAM cells having isolated read and write paths that allow read and write currents to flow simultaneously through the same bit-cell without affecting each other's functionality. The control circuit dynamically adjusts the write current value based on the number of cells being written. It is designed to handle the maximum number of inputs per clock cycle, enhancing parallelism by activating the corresponding word lines efficiently by using an input encoding design unit. It also manages multiple weight gradient calculations along with multiple weight updates. The read outputs are then conveyed on ADCs, which are then added together in parallel with the help of a local summation unit. Then, the sum of all local summation units is consolidated by the global summation unit.

C. SOT-MRAM based Multi-Port Bit-Cell Design

To enhance the energy-efficiency and parallelism in CIM architecture, a multi-port SOT-MRAM bit-cell design is proposed. The schematic of the proposed multi-port SOT-MRAM based bit-cell is shown in Fig. 4 (a). Please note that the write current in SOT-cell does not pass through the SOT device layer stacks, it flows through a heavy metal layer instead, as shown in Fig. 1. This ensures that the write current for both write-0 and write-1 operations remains the same at the cell level. In contrast, other two-terminal memristor devices such as ReRAM, *Spin Transfer Torque MRAM* (STT-MRAM), and *Phase-Change Memory* (PCM) exhibit asymmetry, as the current passes through the device itself, and its resistance value creates this imbalance [29], [30]. To address this asymmetry beyond the bit-cell level, we have incorporated an additional transistor (N2) into the write path of the conventional bit-cell design, as illustrated in Fig. 4 (a) [31].

During a write operation, both *Write Word Lines* (WWL1 and WWL2) are set to 1, turning ON transistors N1 and N2, while *Read Word Lines* (RWLs) are set to 0, keeping N3-N8 OFF. The sizes of the N1 and N2 are kept large to drive a high current for writing. The *Source Line* (SL) and *Word Bit Line* (WBL) are kept either 1 or 0 depending upon the bit to be written. The write path is thus established between N1 and

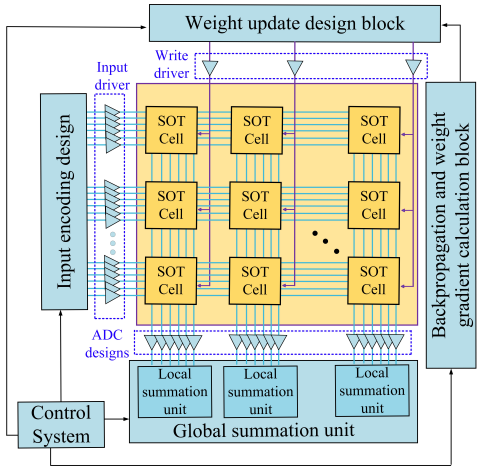
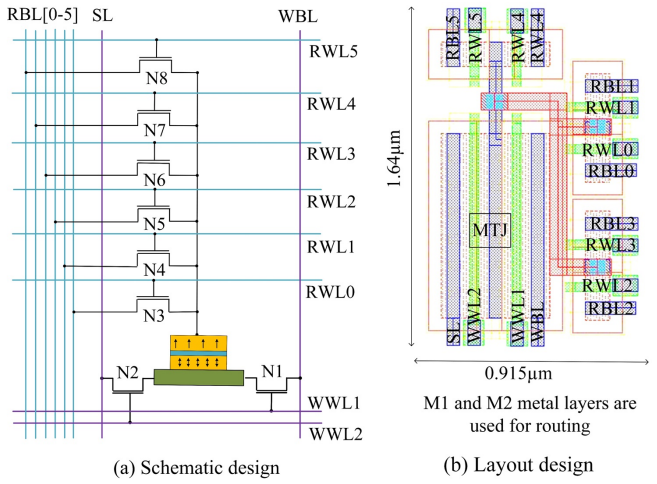


Fig. 3. Macro-level architecture of proposed single tile.



(a) Schematic design (b) Layout design

Fig. 4. Illustration of one-write six-read multi-port SOT-MRAM bit-cell.

N2. For a read operation, both SL and WBL are set to 0. To establish the read path, N2 is kept ON by setting WWL1 to 1 and WWL2 to 0, while RWLs are set to 1 whenever the read transistors need to be accessed. In contrast to the 2T1R bit-cell in Fig. 1, the two WWLs are used so that only one transistor out of N2 or N1 is set ON during the read operation. To sense the output, the sense amplifier is connected to the *Read Bit Line* (RBL) at one end and reference at another. The layout of a one-write and six-read bit-cell design, shown in Fig. 4 (b), having one write port and six read ports, implemented using *Generic Process Design Kit* (GPDK) 45 nm technology and utilizing two metal layers.

D. Design Implementation Strategy

1) Bulk write mechanism for weight updates: As we know, weight updates are a crucial step where the computing engine adapts itself for real-time applications using an on-chip training mechanism. In our proposed architecture, we perform these updates for multiple cells simultaneously using a bulk-write mechanism, as shown in Fig. 5. After backpropagation and weight gradient calculation, the new weights are stored in memory, and the write 0's and 1's to be updated are identified. We first select the 1's, store their locations in latches, then pass a high current using write drivers, selecting those to be written based on the latch values. Since the write-1 operation shares the current in the same direction and it has symmetric current paths, so it is easily managed using write drivers.

We have conducted a cell-level analysis to determine the effective current (I_e) at which energy consumption is minimum that is required for a successful write thus, optimizing energy-efficiency (see Fig. 7 (a)). The maximum current provided by the write driver depends on both the technology and design. Therefore, the number of cells (N) that can be written in parallel is the ratio of the total current (maximum) to the effective current (I_e) for writing a single cell. If the number of cells exceeds N, additional iterations are needed to complete the write-1 operation. Here, the control circuit dynamically adjusts the current based on the number of cells, ensuring energy-efficient writing. In this way, weights are mapped to the conductance values of SOT-MRAM bit-cells across columns in the crossbar array in parallel while maintaining energy-efficiency. Once the write-1 process is complete, we perform the write-0 operation, following a similar procedure but reversing the current direction.

2) Design for inference: In CIM architectures, inference operations typically involve applying input vectors to crossbars, which represent matrices, in order to perform vector-matrix multiplications. In our proposed architecture, multiple vectors are applied to the crossbars through a multi-port cell design, as illustrated in Fig. 6. The vector elements are connected to the gates of the access transistors. When a read voltage is applied through the RBL, a dot product operation is performed, and the result is quantized using an ADC. We employ a five-bit precision SAR ADC design in this work, as described by [32]. Based on our cell-level analysis, we can perform up to six sets of MAC operations (for example, executing six separate vector-matrix multiplications in parallel) with our multi-port bit-cell design while maintaining a margin of 40 mV [14]. Furthermore, once the values for all vectors are obtained, they are combined to generate the final output for the neural network.

IV. Results

In this section, we provide a detailed description of the setup used for calculating results, followed by a comprehensive explanation of both circuit-level and system-level evaluation.

A. Setup

For the circuit-level implementation, we employed the SOT-MRAM based model proposed in [33] with parameters depicted in Table I. We employed a GPDK 45 nm SPICE model to simulate the CMOS components and design the layout. The simulations are performed using the Cadence

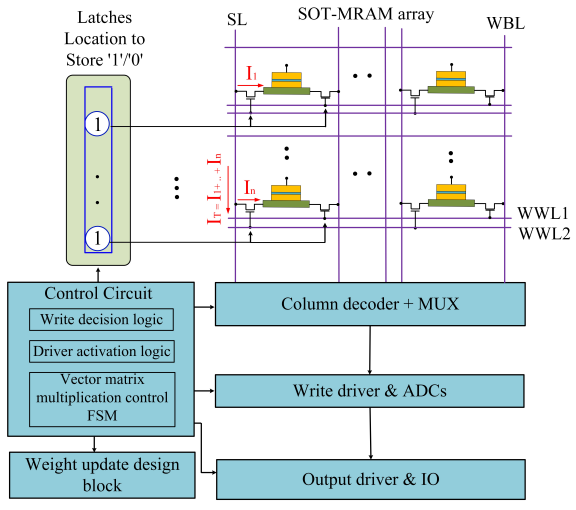


Fig. 5. Weight update implementation using bulk write mechanism.

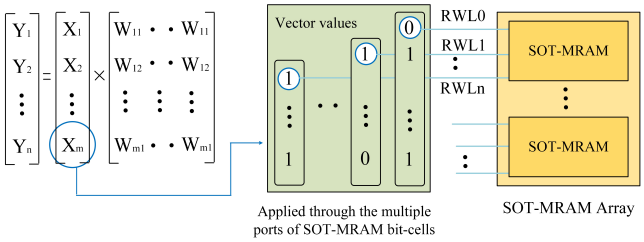


Fig. 6. Multiple vector multiplication for inference using multi-port design.

Virtuoso with a supply voltage of 1.2 V and temperature of 27°C as well as process variation of $\pm 3\sigma$ is considered. We used Cadence Genus for synthesis and Calibre PEX for parasitic extraction, while also considering the worst-case scenarios for the target cell, row, and column. For system-level implementation, we used the VGG-8 architecture to classify 32×32 color images from the CIFAR-10 dataset [34], [35]. This architecture consists of six convolutional layers followed by two fully connected layers to perform the classification. Each convolution layer is followed by a max pooling layer. The DNN + Neurosim V2.0 framework is utilized to evaluate CIM inference and training performance, which facilitates software/hardware co-design [26].

B. Circuit-Level Evaluation

In SOT-MRAM, the switching delay depends upon the amount of current flowing through the bit-cell, as the current increases a shorter switching delay is obtained. The cell is symmetric, so for writing both 1 and 0, the same delay is obtained. Fig. 7 (a) shows the energy consumption for various current overdrive values (I_e/I_c, where I_e is the effective current value at which energy consumption is minimum (159 μA) and I_c is the critical current (128 μA). As illustrated, the energy consumption is minimized for a current overdrive factor of 1.24 value. Using the above I_e value, the number

TABLE I
SOT-MRAM PARAMETERS

Parameters	Value
Saturation Magnetization	1×10^6 A/m
Damping Factor	0.08
Spin Hall Angle	0.21
Critical Current	128 μA
Free Layer Dimensions	150 nm × 150 nm × 0.6 nm
Heavy Metal Dimensions	200 nm × 150 nm × 3 nm

TABLE II
PERFORMANCE OF VARIOUS COMPONENTS AND OPERATIONS FOR STANDARD AND PROPOSED (SIX WRITE AND SIX READ) CIM ARCHITECTURE USING 128 × 128 ARRAY

	Standard SOT-CIM		Proposed SOT-CIM	
Components	Latency (s)	Energy (J)	Latency (s)	Energy (J)
Accumulation	5.32	0.22	4.31	0.15
WG Computation	17.63	1.89	10.08	0.62
Feed-Forward	17.79	0.53	13.61	0.30
Error in BP	14.89	0.77	10.63	0.48
Weight Gradient	24.50	21.39	15.01	15.25
Weight Update	0.72	0.015	0.02	0.004

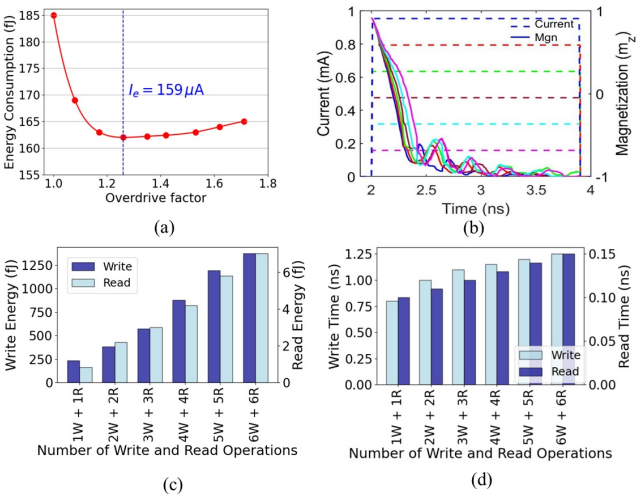


Fig. 7. (a) Energy consumption versus overdrive factor for a single bit-cell, (b) Illustration of bulk-write: current flow and switching behavior, (c) Write and read energy (d) Write and Read timings for multi-port bit-cell

of MRAMs that can be written in an energy-efficient way is calculated to be six, considering the maximum current the technology can handle is 1 mA. Fig. 7 (b) illustrates the current and switching behavior of six SOT-MRAM cells, with a maximum current of 953 µA required to ensure successful switching of all six MTJs. For the read operation, six ports were chosen because adding another port results in incorrect read values due to process variations and reduced read margins, making it unreliable for data retrieval. Additionally, Fig. 7 (c) and (d) show the performance of the proposed approach in terms of energy and time measurements for writing to the cell and reading from the cell using standard one write + one read and proposed six write + six read operations, respectively.

C. Crossbar Sub-System Evaluation

The synaptic crossbar array of size 128 × 128 has been implemented on the Cadence Virtuoso for each layer separately. Each column's output is fed to an output neuron. The results in terms of latency and energy consumption are measured over 256 epochs having 13M parameters and 0.7M operations. Table II shows the detailed hardware estimation results for latency, energy, and area consumption by main components and operations (feed-forward, error in backpropagation, weight gradient, and weight update) in the CIM accelerator. Fig. 8 (a) and (b) presents the energy and latency consumption for various components respectively, such as ADC, accumulation, weight update, and gradient calculation. Latency and energy consumption are mainly contributed by feed-forward, error computation, gradient computation, and weight update. Among these, weight gradient computation dominates total energy and latency due to the multiple read and write operations involving error and activation functions to compute gradients. It is observed that weight updates utilize less computation energy and latency compared to other computations because they are performed only once per batch.

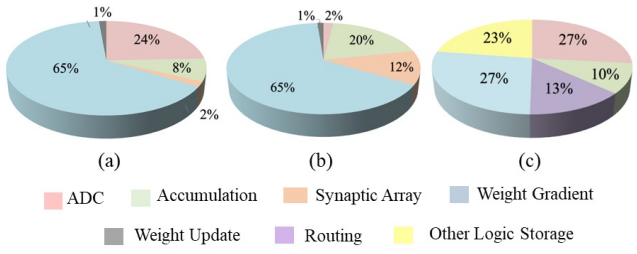


Fig. 8. Breakdown results of the proposed circuit for (a) Energy, (b) Latency, and (c) Area by main components for 128 × 128 array.

Fig. 8 (c) shows the area utilized by the crossbar array, routing, ADCs, accumulation module, weight gradient storage array, and other peripheral circuitry. The ADC and weight gradient computation occupy a large area because these units are made up of conventional CMOS technology and are separate units from CIM arrays. Other logic and storage units occupy comparable areas for data transfer. The accumulation logic includes processing elements, adder trees, and gradient units.

Fig. 9 provides the system-level comparison of latency, area, energy, and throughput for reading and writing parallelly. The latency is decreased as multiple cells are accessed in parallel simultaneously. The throughput is increased due to high parallelism, which allows simultaneous processing of multiple vector elements. This parallelism reduces the number of cycles needed for computations and thus decreases the overall energy per inference. With each additional port, more operations can be performed concurrently, resulting in lower energy consumption per operation. The benefits of a multi-port design come at the cost of increased area due to additional circuitry to manage the parallel access. It is observed that SOT-MRAM based multi-port neural networks are 5.82× and 3.20× more efficient in terms of latency and energy-efficiency, respectively compared to single port architecture. The drawback of a multi-port architecture is higher area consumption. The tile area of a multi-port configuration is increased by 2.67× compared to a single-port configuration.

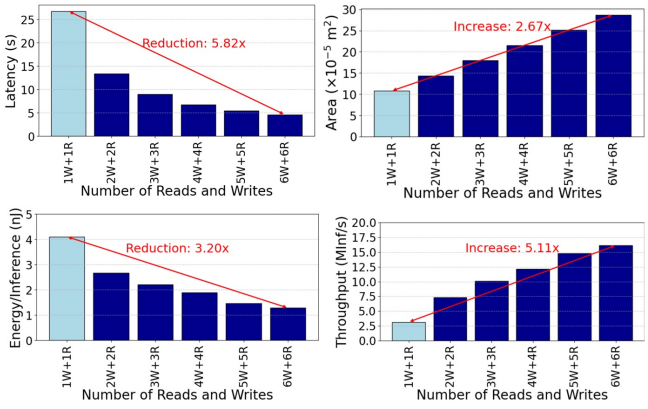


Fig. 9. System-level comparison between standard SOT cell (1R1W) and various multi-ported configuration cell designs. [R = Read, W = Write]

TABLE III
SYSTEM-LEVEL PERFORMANCE OF DIFFERENT CIM ARCHITECTURES FOR ON-CHIP LEARNING

	[26]	[36]	[12]*	Proposed
Technology (nm)	32	7	45	45
Device	SRAM	SOT	SOT	SOT
Area (mm^2)	138.95	10.80	31.67	50.79
Energy-Efficiency (TOPS/W)	-	46.10	20.28	47.15
Throughput (TOPS)	-	-	0.61	3.17
Training Energy-Efficiency (TOPS/W)	1.94	-	2.00	3.49
Training Throughput (TOPS)	0.78	-	0.85	2.04

* Values extracted from DNN + Neurosim V2.0 using bit-cell design in [12]

D. Comparison with State-of-the-Art

Table III compares the system-level performance of different CIM architectures for *Deep Neural Network* (DNN). The area occupied by the proposed architecture increases to 50.79 mm² due to the addition of multiple ports. Despite the larger area, the proposed work achieves energy-efficiency of 47.15 TOPS/W, closely matching the 7 nm technology of 46.10 TOPS/W due to the increased parallelism enabled by multi-port design, which enhances throughput and reduces energy per operation. Notably, the throughput is significantly increased in the proposed work, with a value of 3.17 TOPS, much larger than the 0.61 TOPS evaluated in standard SOT-MRAM. The training energy-efficiency of the proposed SOT-MRAM based CIM architecture is improved to 3.49 TOPS/W compared to 2.00 TOPS/W for standard SOT-MRAM bit-cell. The proposed architecture significantly increases training throughput to 2.04 TOPS, compared to 0.78 TOPS in SRAM and 0.85 TOPS in standard SOT-MRAM.

V. CONCLUSION

This paper presents a novel SOT-MRAM based CIM architecture aimed at enhancing the on-chip learning and inference process. The architecture utilizes multi-port memory to significantly boost parallelism, along with a control circuit that optimizes input requests to maximize port utilization and handles the maximum number of inputs. The results demonstrate a substantial reduction in latency by 5.82× and an increase in energy-efficiency by 3.20× compared to standard single-port designs. These findings highlight the potential of the proposed architecture to significantly improve latency, and increase energy-efficiency for on-chip learning in AI and machine learning applications.

REFERENCES

[1] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and memory wall," IEEE Micro, vol. 44, no. 3, pp. 33-39, 2024.

[2] A. Sehgal, S. Dhull, S. Roy, and B. K. Kaushik, "Advancements in memory technologies for artificial synapses," Journal of Materials Chemistry C, vol. 12, pp. 5274-5298, 2024.

[3] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal, "In-memory computing in emerging memory technologies for machine learning: An overview," ACM/IEEE Design Automation Conference (DAC), pp. 1-6, 2020.

[4] F. H. Meng and W. D. Lu, "Compute-in-memory technologies for deep learning acceleration," IEEE Nanotechnology Magazine, vol. 18, no. 1, pp. 44-52, 2024.

[5] F. Oboril, R. Bishnoi, M. Ebrahimi, and M. B. Tahoori, "Evaluation of hybrid memory technologies using SOT-MRAM for on-chip cache hierarchy," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 3, pp. 367-380, 2015.

[6] S. Diware, S. Dash, A. Gebregiorgis, R. V. Joshi, C. Strydis, S. Hamdioui, and R. Bishnoi, "Severity-based hierarchical ECG classification using neural networks", IEEE Transactions on Biomedical Circuits and Systems (TBioCAS) , vol. 17, pp. 77-91, 2023.

[7] S. Diware, A. Singh, A. Gebregiorgis, R. V. Joshi, S. Hamdioui, and R. Bishnoi, "Accurate and energy-efficient bit-slicing for RRAM-based neural networks", IEEE Transactions on Emerging Topics in Computational Intelligence (TETCI), vol. 7, pp. 164-177, 2022.

[8] V. D. Nguyen, S. Rao, K. Wostyn, and S. Couet, "Recent progress in spin-orbit torque magnetic random-access memory," NPJ Spintronics, vol. 2, no. 1, pp. 48, 2024.

[9] D. C. Worledge and G. Hu, "Spin-transfer torque magnetoresistive random access memory technology status and future directions," Nature Reviews Electrical Engineering, vol. 1, pp. 730-747, 2024.

[10] G. Prenat, K. Jabeur, P. Vanhauwaert, G. D. Pendina, F. Oboril, and R. Bishnoi, "Ultra-fast and high-reliability SOT-MRAM: From cache replacement to normally-off computing," IEEE Transactions on Multi-Scale Computing Systems, vol. 2, no. 1, pp. 49-60, 2016.

[11] R. Bishnoi, F. Oboril, and M. B. Tahoori, "Non-volatile non-shadow flip-flop using spin orbit torque for efficient normally-off computing," 21st Asia and South Pacific Design Automation Conference (ASP-DAC), Macao, China, pp. 769-774, 2016.

[12] Y. Luo, P. Kumar, Y. C. Liao, W. Hwang, F. Xue, W. Tsai, S. X. Wang, A. Naeemi, and S. Yu, "Performance benchmarking of spin-orbit torque magnetic RAM (SOT-MRAM) for deep neural network accelerators," Design and Applications of Emerging Computer Systems, pp. 67-89, 2023.

[13] Q. Liu, B. Gao, P. Yao, D. Wu, J. Chen, Y. Pang, W. Zhang, Y. Liao, C. Xue, W. Chen, J. Tang, Y. Wang, M. Chang, H. Qian, and H. Wu, "33.2 A fully integrated analog ReRAM based 78.4 TOPS/W compute-in-memory chip with fully parallel MAC computing," IEEE International Solid-State Circuits Conference (ISSCC), pp. 500-502, 2020.

[14] A. Singh, R. Bishnoi, R. V. Joshi, and S. Hamdioui, "Referencing-in-array scheme for RRAM-based CIM architecture," Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1413-1418, 2022.

[15] J. Wu, X. Huang, L. Yang, J. Wang, B. Liu, Z. Wen, J. Li, G. Yu, K. S. Chong, and C. Wang, "An energy-efficient deep belief network processor based on heterogeneous multi-core architecture with transposable memory and on-chip learning," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 11, no. 4, 2021.

[16] L. Huijbregts, H. H. Liu, P. Detterer, S. Hamdioui, A. Yousefzadeh, and R. Bishnoi, "Energy-efficient SNN architecture using 3nm FinFET multiport SRAM-based CIM with online learning," Proceedings of the 61st ACM/IEEE Design Automation Conference, pp. 1-6, 2024.

[17] S. Yu, H. Jiang, S. Huang, X. Peng, and A. Lu, "Compute-in-memory chips for deep learning: Recent trends and prospects," IEEE Circuits and Systems Magazine, vol. 21, no. 3, pp. 31-56, 2021.

[18] S. Diware, K. Chilakala, R. V. Joshi, S. Hamdioui, and R. Bishnoi, "Reliable and energy-efficient diabetic retinopathy screening using memristor-based neural networks", IEEE Access, vol. 12, pp. 47469-47482, 2024.

[19] S. Diware, A. Gebregiorgis, R. V. Joshi, S. Hamdioui, and R. Bishnoi, "Mapping-aware biased training for accurate memristor-based neural networks", IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 1-5, Hangzhou China, 2023.

[20] M. Ali, S. Roy, U. Saxena, T. Sharma, A. Raghunathan, and K. Roy, "Compute-in-memory technologies and architectures for deep learning workloads," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 11, 2022.

[21] Y. Kim, X. Fong, K.-W. Kwon, M.-C. Chen, and K. Roy, "Multilevel spin-orbit torque MRAMs," IEEE Transactions on Electron Devices, vol. 62, no. 2, pp. 561–568, 2015.

[22] Z. He, S. Angizi, F. Parveen, and D. Fan, "High performance and energy-efficient in-memory computing architecture based on SOT-MRAM," IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), pp. 97-102, 2017.

[23] L. Liu, C. F. Pai, Y. Li, H. W. Tseng, D. C. Ralph, and R. A. Buhrman, "Spin-torque switching with the giant spin hall effect of tantalum," Science, vol. 336, no. 6081, pp. 555-558, 2012.

[24] T. Endoh, H. Honjo, K. Nishioka, and S. Ikeda, "Recent progresses in STT-MRAM and SOT-MRAM for next generation MRAM," IEEE Symposium on VLSI Technology, pp. 1-2, 2020.

[25] I. M. Miron, K. Garello, G. Gaudin, P. J. Zermatten, M. V. Costache, S. Auffret, S. Bandiera, B. Rodmacq, A. Schuhl, and P. Gambardella,"Perpendicular switching of a single ferromagnetic layer induced by in-plane current injection," Nature, vol. 476, no. 7359, pp. 189–193, 2011.

[26] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "DNN+Neurosim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 11, pp. 2306-2319, 2021.

[27] J. Bilski, B. Kowalczyk, A. Marjański, M. Gandor, and J. Zurada, "A novel fast feedforward neural networks training algorithm," Journal of Artificial Intelligence and Soft Computing Research, vol. 11, no. 4, pp. 287-306, 2021.

[28] Z. Zhang, X. Liu, H. Zhou, S. Xu, and C. Lee, "Advances in machine-learning enhanced nanosensors: From cloud artificial intelligence toward future edge computing at chip level," Small Structures, vol. 5, pp. 2300325, 2023.

[29] N. K. Upadhyay, H. Jiang, Z. Wang, S. Asapu, Q. Xia, and J. J. Yang, "Emerging memory devices for neuromorphic computing," Advanced Materials Technologies, vol. 4, pp. 1800589, 2019.

[30] Y. Kim, S. K. Gupta, S. P. Park, G. Panagopoulos, and K. Roy, "Write-optimized reliable design of STT MRAM," ACM/ IEEE International Symposium on Low Power Electronics and Design, pp. 3-8, 2012.

[31] R. Bishnoi, F. Oboril, and M. B. Tahoori, "Low-power multi-port memory architecture based on spin-orbit torque magnetic devices," Great Lakes Symposium on VLSI, pp. 409-414, 2016.

[32] H. Jiang, W. Li, S. Huang, S. Cosemans, F. Catthoor, and S. Yu, "Analog-to-digital converter design exploration for compute-in-memory accelerators," IEEE Design and Test, vol. 39, no. 2, pp. 48-55, 2022.

[33] K. Kazemi, G. E. Rowlands, E. Ipek, R. A. Buhrman, and E. G. Friedman, "Compact model for spin–orbit magnetic tunnel junctions," IEEE Transactions on Electron Devices, vol. 63, no. 2, pp. 848-855, 2016.

[34] K. Simonyan, and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," ArXiv Preprint, arXiv:1409.1556, 2014.

[35] R. Doon, T. Kumar Rawat, and S. Gautam, "Cifar-10 classification using deep convolutional neural network," IEEE Punecon, pp. 1-5, 2018.

[36] J. Doevenspeck, K. Garello, B. Verhoef, R. Degraeve, S. Van Beek, D. Crotti, F. Yasin, S. Couet, G. Jayakumar, I. A. Papistas, P. Debacker, R. Lauwereins, W. Dehaene, G.S. Kar, S. Cosemans, A. Mallik, and D. Verkest, "SOT-MRAM based analog in-memory computing for DNN inference," IEEE Symposium on VLSI Technology, pp. 1-2, 2020.

All-in-Memory Stochastic Computing using ReRAM

João Paulo C. de Lima[1,2], Mehran Shoushtari Moghadam[3],
Sercan Aygun[4], Jeronimo Castrillon[1,2,5], M. Hassan Najafi[3], and Asif Ali Khan[1]
[1]Chair for Compiler Construction, Dresden University of Technology, Dresden, Germany
[2]Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), Dresden, Germany
[3]Electrical, Computer, and Systems Engineering Department, Case Western Reserve University, Cleveland, OH, USA
[4]School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA
[5]Barkhausen Institut, Dresden, Germany
Corresponding authors: {joao.lima, asif_ali.khan}@tu-dresden.de

***Abstract*—As the demand for efficient, low-power computing in embedded and edge devices grows, traditional computing methods are becoming less effective for handling complex tasks. Stochastic computing (SC) offers a promising alternative by approximating complex arithmetic operations, such as addition and multiplication, using simple bitwise operations, like `majority` or `AND`, on random bit-streams. While SC operations are inherently fault-tolerant, their accuracy largely depends on the length and quality of the stochastic bit-streams (SBS). These bit-streams are typically generated by CMOS-based stochastic bit-stream generators that consume over 80% of the SC system's power and area. Current SC solutions focus on optimizing the logic gates but often neglect the high cost of moving the bit-streams between memory and processor. This work leverages the physics of emerging ReRAM devices to implement the entire SC flow in place: ❶ generating low-cost true random numbers and SBSs, ❷ conducting SC operations, and ❸ converting SBSs back to binary. Considering the low reliability of ReRAM cells, we demonstrate how SC's robustness to errors copes with ReRAM's variability. Our evaluation shows significant improvements in throughput (1.39×, 2.16×) and energy consumption (1.15×, 2.8×) over state-of-the-art (CMOS- and ReRAM-based) solutions, respectively, with an average image quality drop of 5% across multiple SBS lengths and image processing tasks.**

I. Introduction

The growing prevalence of embedded and edge devices has driven the demand for low-cost but efficient computing solutions. These devices, which often run complex applications like computer vision tasks in real-world environments, are constrained by computational resources and power budget, making traditional computing methods less effective. Stochastic computing (SC) and non-von Neumann paradigms have emerged as promising alternatives, offering trade-offs in computational density, energy efficiency, and error tolerance [1, 2, 3, 4].

In SC, data is represented by random bit-streams, where a value $x \in [0, 1]$ is encoded by the probability (P_x) of a '1' appearing in the stream. For example, the bit-stream '10101' represents the value $\frac{3}{5}$, where 5 is the bit-stream length (N). This unconventional representation enables complex computations like *multiplication* and *addition* to be approximated with simple logic operations such as `AND` and `majority`, respectively, reducing area and power consumption substantially without taking a high toll on computational accuracy. Additionally, since all bits in the bit-streams carry equal weights – no *most*- or *least*-significant bits – SC is naturally tolerant to noise, including bit flips and inaccuracies in the input data and computations. This makes SC particularly advantageous for a range of applications, including image processing [5], signal processing [6], and neural networks [7].

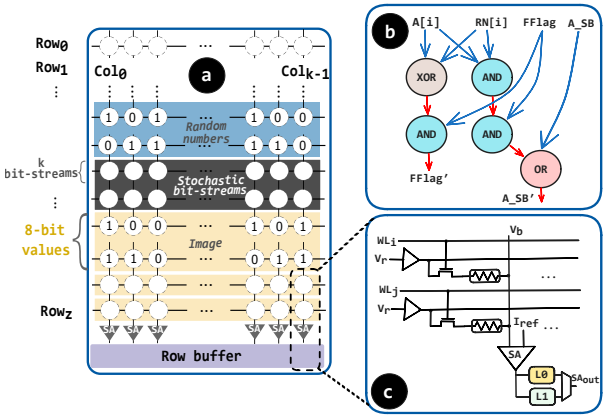


Fig. 1. A high-level overview of our proposed in-memory SC solution: (a) ReRAM array, (b) Greater-than operation using basic logic gates, (c) Write latches in the peripheral circuitry.

Stochastic bit-streams (SBSs) are conventionally generated using a CMOS-based structure, called stochastic bit-stream generator, built from a *pseudo random* (or more recently, *quasi-random)* [8] *number generator*, and a *binary comparator*. The accuracy and cost efficiency of SC systems highly depend on this bit-stream generation unit. Presently, CMOS-based bit-stream generation consumes up to 80% of the system's total hardware cost and energy consumption [4, 9]. Additionally, SC implementations on classic von Neumann systems require extensive movement of bit-streams *from/to* memory, which is often overlooked in evaluations but can easily offset the benefits of the simpler SC operations. This has motivated significant research into non-von Neumann computing for SC, using different memory technologies [10, 11, 12, 13, 14, 15, 16, 17].

Most compute-in-memory (CIM) designs for SC utilize ReRAM, a nonvolatile memory technology that stores data in the resistance state of the devices [18]. ReRAM offers DRAM-comparable read latency, higher density, and significantly reduced read energy consumption but incurs expensive write operations, limited write endurance, and suffers from non-linearities [19]. SC in ReRAM benefits from ReRAM's high density for efficient storage and in-place processing of long bit-streams, while SC's inherent robustness helps mitigate the effects of ReRAM's non-linearities. In existing ReRAM-based CIM designs for SC, conventional CMOS-based random

number generators (RNGs) are used for bit-stream generation, while ReRAM arrays handled in-place logic operations [10, 11, 12]. This increases parallelism and reduces data movement overhead, though the high cost of random bit-stream and SBSs generation remains a bottleneck. Techniques like leveraging ReRAM's inherent write noise [13] and employing DRAM-based lookup tables [20] have been explored to improve the performance of random bit-stream generation. However, SBSs generation continues to face challenges, including the high energy cost of ReRAM write operations and scalability issues with DRAM-based methods.

To address these challenges, we propose a CIM accelerator that implements *all steps* of SC using ReRAM. We decouple RNG from SBS generation, allowing compatibility with any RNG type, including general-purpose true RNGs (TRNGs) based on ReRAM [21]. This approach ensures (1) *accurate SBS generation* with target probabilities and (2) *correlation control*, despite ReRAM cell noise. We perform in-memory logic and comparison operations to convert true random numbers into SBS (❶), conduct SC operations (❷), and convert the resulting values back into a binary representation (❸). Concretely, this work makes the following novel contributions:

- We propose a ReRAM-based accelerator for SC that implements all steps in place, including those often overlooked by current SC designs.
- We develop a novel in-memory method for converting true random binary sequences (50% *ones*) into SBSs with desired probabilities. To the best of our knowledge, this is the first such method reported in the literature.
- Our SBS generation approach is RNG-agnostic, leveraging in-memory comparison to produce SBS even under substantial CIM failures caused by ReRAM variability.
- For SC operations typically implemented with `MUX`s, we propose novel alternatives that are CIM-friendly and achieve comparable accuracy.

Compared to the state-of-the-art CMOS-based solutions, the proposed design, while requiring minimal changes to the memory periphery, on average, reduces energy consumption by $1.15\times$ and improves throughput by $1.39\times$ across multiple image processing applications. Our design is also more robust than traditional arithmetic for CIM, with only a 5% average quality drop in the presence of faults compared to a 47% drop. It eliminates the need for protection schemes on unreliable ReRAM devices and provides better correlation control than previous in-memory SC designs.

II. Background and Related Work

A. ReRAM-based Computing

Resistive RAM (ReRAM) is a type of nonvolatile memory where each cell, typically a metal oxide, and being referred to as a memristor, is programmed to different resistance states using an electric voltage [22]. Data is represented through resistance levels, such as a high resistance state (HRS) for '0' and a low resistance state (LRS) for '1'. Organized in a conventional memories like 2D grid of rows (wordlines, WL) and columns (bitlines, BL), ReRAM promises DRAM-comparable read performance but has costly write operations that impact both energy consumption and the write endurance [19]. For CIM using ReRAM, the 1T1R (one transistor, one resistor) crossbars are extensively used in machine learning and other domains to perform analog matrix-vector multiplication in constant time [3]. Similarly, stateful and non-stateful logic techniques, such as MAGIC [23] and scouting logic (SL) [24], respectively, have been demonstrated for implementing logic operations using ReRAM. ReRAM cells have inherent stochasticity and noise; which have also been investigated to generate true random numbers [21, 25].

B. Stochastic Computing (SC)

SC is an alternate computing approach offering simple execution of complex arithmetic operations and high tolerance to soft errors. Unlike traditional binary radix, SC operates on random bit-streams of '0's and '1's, with no bit-significance. SC systems include three primary components: ❶ Bit-stream generator that converts data from traditional binary to stochastic bit-stream, ❷ computation logic that performs bit-wise operations on the bit-streams, and ❸ bit-stream to binary converter to convert data back to binary format.

Bit-stream Generation: The accuracy of SC operations highly depends on the quality of bit-streams. To convert a binary number X to an SBS of size N, an RNG is used to generate N random numbers. The SBS is generated by comparing each of these N random numbers with X. A '1' is produced if the random number is less than X, and a '0' is produced otherwise. Conventionally, SC systems employ CMOS-based pseudo-RNGs (PRNGs) such as linear-feedback shift registers (LFSRs) to generate the needed random numbers [26]. However, this can lead to suboptimal performance as very long SBSs are needed to achieve acceptable accuracy. Recent works leverage quasi-RNGs (QRNGs) for better accuracy [27] but at the cost of a higher area and power [8, 9]. The high cost of CMOS-based SBS generation offsets the gains made with simple computation circuits.

SC Operations: Basic arithmetic operations – multiplication, addition, subtraction, and division – are implemented in SC using minimal components: an `AND` gate, a multiplexer (`MUX`) unit, an `XOR` gate, and a `MUX+D`-flip-flop, respectively (Fig. 2) [4, 28]. For N-bit-long SBSs, the logic operations are often performed serially, producing the output SBS in N clock cycles. Parallel execution of the operations is also feasible by trading off time with space. This approach is particularly attractive for SC with CIM as it enables fast and independent execution of all bit-wise operations. For correct functionality of the aforementioned operations, the input bit-streams must provide the desired correlation level, i.e., *uncorrelated* for the multiplication and addition, and *correlated* for the subtraction and division operations. The independence (i.e., uncorrelation) requirement is typically satisfied by using different RNGs while the desired amount of correlation is guaranteed by using shared RNGs when generating SBSs.

C. State-of-the-art In-memory SC Solutions

Existing CIM-SC designs are mostly hybrid, i.e., either memristive arrays are used to generate random numbers and CMOS logic to perform computations, or vice versa. For instance, Knag et al. [10] proposed generating SBSs using memristors and off-memory computations using CMOS

stochastic circuits. A similar design is proposed in [16] that exploits the switching stochasticity of probabilistic Conductive Bridging RAM (CBRAM) devices to generate SBSs in memory efficiently. The generated bit-streams are then used to optimize deep learning parameters using a hybrid CMOS-memristor stochastic processor. ReRAM-based SBS generation has also been proposed for SC. However, these solutions primarily use the probabilistic switching, i.e., the write operation in ReRAM [29]. Riahi Alam et al. [17] developed an accurate method for in-memory SC multiplication by performing a deterministic binary-to-SBSs conversion. Sun et al. [15] employed unary coding, using multi-level memristor cells, for weight representation in a ReRAM-based neural network accelerator. The most relevant work to our design is SCRIMP [13], which also proposes SBS generation and computation using ReRAM. However, similar to other prior works, it generates SBSs using the stochasticity in the write operation, which is not only extremely slow but also affects write endurance. Existing methods can generate SBSs with target probabilities but *lack correlation control*, which limits their applicability for SC operations that require correlated inputs. We propose a novel ReRAM-based solution to convert true random binary sequences (50% *ones*) into SBSs with desired probabilities using bulk-bitwise operations.

III. Proposed In-ReRAM Stochastic Computing

We exploit the physical properties of ReRAM arrays to implement all stages of the SC flow (see Sec. II-B). In practice, we use multiple arrays to parallelize and pipeline the different stages. However, for simplicity, we show a single array in Fig. 1ⓐ, consisting of dedicated rows to store input binary data (yellow), random numbers (blue), and in-memory generated stochastic bit-streams (grey). In the following, we explain the in-memory implementation of these different operations and the data flow in the different stages of the SC flow.

A. Stochastic Number Generation (SNG)

The switching stochasticity of ReRAM devices has been exploited to generate true random numbers (see Sec. II-A). We build upon this prior work and consider TRNG as a single-step operation that stores random sequences directly in ReRAM arrays. To generate SBSs from these random sequences (referred to as in-memory SNG or IMSNG), we compare them with n-bit input binary operands using in-memory bitwise operations (see Sec. II-B). Concretely, for comparing two binary numbers A and RN in memory, starting from the most significant bit (MSB) to the least significant bit (LSB), we perform bitwise comparison and stop at the first non-equal bit position, i.e., where $A_i \oplus RN_i$ is 1. This is achieved by implementing the `greater-than` operation, i.e., $A_i > RN_i$, using in-ReRAM bitwise `XOR` and `AND` operation, together with a flag bit (`FFlag`), as illustrated in the Boolean network in Fig. 1ⓑ. The result of this comparison is a row A_{SB} representing the SBS of A. This network is converted into data structures like `XOR-AND`-Inverter graph (XAG) for manipulation and optimization using logic synthesis tools [30].

Considering the SL approach (see Sec. II-A), implementing this network requires $5n$ operations, as each logic gate requires one sensing step (cf. [24]). This also requires writing the intermediate signals (e.g., that of `XOR`) back to the ReRAM array at least 4 times (red arrows in Fig. 1ⓑ). These intermediate writes can be avoided by forwarding the output of one operation directly as input voltage to the bitline for the next operation. This requires the periphery to incorporate a simple feedback mechanism that converts the latched signal to adjust the bitline voltage (V_b) (see Fig. 1ⓒ) to mimic the voltage drop across the ReRAM that would have been written. This approach (referred to as *IMSNG-naive*) reduces the number of ReRAM writes to $2n$.

IMSNG-opt: As an alternative approach, we are exploiting the latches in the ReRAM peripheral circuitry, shown in Fig. 1ⓒ (L0 and L1), to minimize the write overhead. ReRAM and other nonvolatile memories, typically employ double latches and a write driver to conduct differential writes [31]. For each write driver, a latch stores the data to be written to the cells and a second latch stores whether the cell should be modified (in case the new data is different than the already stored data). We leverage this mechanism to directly implement the `AND` operations involving `FFlag` as a predicated sensing, hence eliminating the need to write the intermediate result to the memory cells. This approach, which uses existing latches and drivers to eliminate write operations deriving from intermediate results, is referred to as *IMSNG-opt*.

Table I compares the mean squared error (MSE) of our IMSNG to a software-based (SW), PRNG-based, and QRNG-based SBS generator. SW uses MATLAB's RNG (`rand`) for bit-stream generation. PRNG and QRNG use 8-bit LFSR and Sobol sequence generators. The comparison in IMSNG is between the target input and N random binary sequences

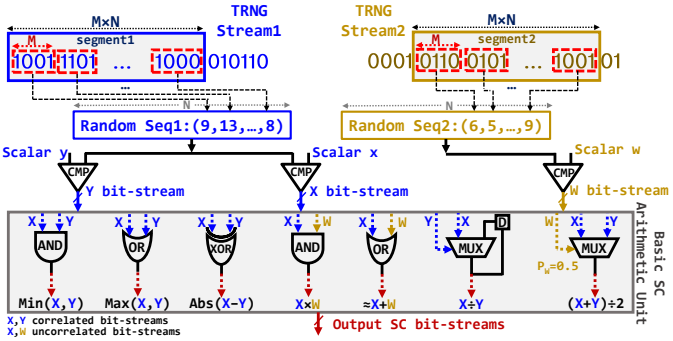


Fig. 2. In-ReRAM SBS generation and SC arithmetic operations. CORDIV [28] is considered for SC division with $X \leqslant Y$. Addition with `OR`; the inputs are in the $[0, 0.5]$ interval to not exceed 1.0 in the output.

TABLE I
MSE(%) COMPARISON OF GENERATING SBSS UTILIZING DIFFERENT RNG SOURCES (M: BLOCK SIZE, N: BIT-STREAM LENGTH).

RNG Source		Bit-stream length (N)				
	Segment size (M)	32	64	128	256	512
IMSNG [21]	5	0.567	0.321	0.189	0.134	0.103
	6	0.562	0.302	0.177	0.114	0.084
	7	0.534	0.279	0.157	0.095	0.064
	8	0.557	0.300	0.177	0.107	0.074
	9	0.520	0.282	0.159	0.090	0.060
Software - MATLAB		0.529	0.264	0.131	0.065	0.032
PRNG (8-Bit LFSR)		1.069	0.554	0.288	0.137	0.071
QRNG (8-Bit Sobol)		0.033	0.008	0.002	5.05×10^{-4}	1.25×10^{-4}

For PRNG, a Maximal length LFSR with polynomial $x^8 + x^5 + x^3 + 1$ is used.

TABLE II
MSE (%) COMPARISON OF SC ARITHMETIC OPERATIONS UTILIZING DIFFERENT RNGS WITH $M = 8$.

SC Operations	IMSNG [21]					Software - MATLAB					PRNG (LFSR)					QRNG (Sobol)				
	N:32	64	128	256	512	N:32	64	128	256	512	N:32	64	128	256	512	N:32	64	128	256	512
Multiplication	0.473	0.255	0.147	0.091	0.061	0.444	0.219	0.108	0.054	0.027	0.851	0.476	0.221	0.093	0.060	0.058	0.017	0.005	0.001	2.9×10^{-4}
Scaled Addition	0.690	0.356	0.193	0.109	0.062	0.648	0.328	0.159	0.082	0.041	1.117	0.607	0.289	0.157	0.065	0.102	0.013	0.003	0.002	2.1×10^{-4}
Approx. Addition	1.548	1.186	1.024	0.927	0.886	1.379	1.055	0.897	0.789	0.751	2.654	1.702	1.180	0.914	0.842	0.463	0.586	0.670	0.662	0.689
Abs. Subtraction	0.641	0.354	0.136	0.144	0.107	0.514	0.263	0.129	0.064	0.034	0.559	0.281	0.136	0.058	0.026	0.016	0.004	0.001	2.5×10^{-4}	6.5×10^{-5}
Division	1.614	0.895	0.518	0.295	0.187	1.454	0.789	0.392	0.196	0.106	2.760	2.140	1.688	1.630	1.477	0.251	0.164	0.129	0.126	0.128
Minimum	0.572	0.307	0.177	0.106	0.064	0.514	0.265	0.130	0.066	0.032	1.493	0.811	0.394	0.199	0.085	0.033	0.008	0.002	5.1×10^{-4}	1.3×10^{-4}
Maximum	0.572	0.302	0.186	0.117	0.077	0.543	0.259	0.132	0.064	0.033	0.481	0.263	0.123	0.073	0.027	0.032	0.008	0.002	5.0×10^{-4}	1.3×10^{-4}

(generated in-memory) of M bits, where $M=5, 6, ..., 9$ (see Fig. 2). The data is based on 1,000,000 samples extracted from a uniform distribution. The results highlight that IMSNG, despite its random fluctuations and true randomness, provides comparable accuracy to other methods. Notably, for bit-stream lengths of 32, 64, and greater than 128, MSEs of approximately 0.5%, 0.3%, and 0.1% are measured, respectively.

B. Stochastic Circuits using Scouting Logic (SL)

SL implements boolean logic using ReRAM read operations with a modified sense amplifier (SA) [24, 32]. During a logic operation, two or more rows are simultaneously activated and the resulting current through the cells in each bitline is compared with a reference current I_{ref} by the SA, whose output is the desired result of the Boolean operation (see Fig. 1c). All basic logic operations such as (N)AND, (N)OR, X(N)OR, and NOT, are realized in a single cycle [33].

Bulk bitwise in-memory operations are performed on large vectors. When operating on traditional binary-radix numbers, we can only exploit the single instruction, multiple data (SIMD) parallelism, since these algorithms are sequential by nature due to carry propagation. In contrast, SC handles basic arithmetic operations (addition, subtraction, multiplication, and division) using simple, low-cost logic units such as AND, NOT, XOR, MUXs, and flip-flops. Each bit is computed independently, allowing for in-memory SC to exploit bulk-bitwise logic and massive word-level parallelism, significantly reducing latency for basic arithmetic operations. In the following, we explain how these primary arithmetic operations are implemented using bulk bitwise logic schemes.

Multiplication is implemented by performing bitwise AND on two independent bit-streams, representing probabilities p and q. The probability of observing a '1' in the output stream equals $p \wedge q$, which aligns well with the principles of SL with the time complexity of $\mathcal{O}(1)$. This contrasts with conventional bulk-bitwise implementations of binary radix multiplication, which exhibit a time complexity of $\mathcal{O}(n^2)$, where n represents the number of bits. This complexity arises from the iterative nature of traditional methods, which rely on bit shifts and additions to compute the product.

Scaled addition is implemented in SC using a 2-to-1 MUX. In SL, a 2-to-1 MUX can be approximated by a CIM-friendly 3-input majority gate (MAJ) [34] that can be computed in a single cycle. For in-place MAJ, a reference current corresponding to the majority of the inputs is required. For instance, to conduct a 3-input MAJ gate operation, we use the same reference current used for the 2-input AND gate, as this detects when at least two out of three inputs are high. The time complexity of our MAJ-based addition is $\mathcal{O}(1)$, which is a significant improvement over both traditional ripple-carry additions in the binary domain and MUX-based addition in existing SC that has a time complexity of $\mathcal{O}(N)$.

Division in prior SC works is implemented using CMOS-based flip-flops and MUXs, and *correlated* bit-streams to approximate $y = \frac{x_1}{x_2}$. In SL, the JK flip-flop's truth table can be implemented using the existing latch-based circuitry (Fig. 1c). The intermediate values from the flip-flop are stored in the existing latch (write driver) and forwarded to the bitline as voltage inputs, eliminating the intermediate write operations and improving energy efficiency and write endurance. This method has a time complexity of $\mathcal{O}(N)$, while existing CIM division methods on integer data [35] require $\mathcal{O}(n^2)$ write cycles.

For **other operations**, such as approximate addition, absolute subtraction, minimum, and maximum, we employ bulk-bitwise operations like OR, XOR, AND, and OR, respectively. In stochastic logic, the reference current for the OR operation is set to detect when at least one of the operands is high.

Table II compares the accuracy of these stochastic operations across different SNG sources. Compared to PRNG, QRNG, and SW-based SNG methods, our IMSNG approach achieves comparable accuracy, and even in some cases (e.g., compared to PRNG-based division) a lower MSE. Still, the important advantage of our method is that it is executed completely in memory, eliminating the overheads of transferring SBSs between memory and processing circuits.

C. Stochastic to Binary Conversion

As a last step in the SC flow, the output of the SC operation needs to be converted back to binary. Existing methods use CMOS counters for stochastic-to-binary (S-to-B) conversion, sequentially counting the '1's in the output bit-stream. In contrast, our approach achieves the count in a single step by using bitline current accumulation. The output bit-stream is applied as input voltages (V_r) to a designated reference column in which all cells have been pre-programmed to low resistance states (see Fig. 1c). The total current through the bitline, representing the population count of the bit-stream, is then measured and digitized using analog-to-digital converters (ADCs).

IV. Evaluation Results and Analysis

The custom SA and the proposed hardware modifications, including the feedback mechanism and latch-based optimizations, were validated with SPICE simulations. Energy consumption and latency metrics of the in-memory design were

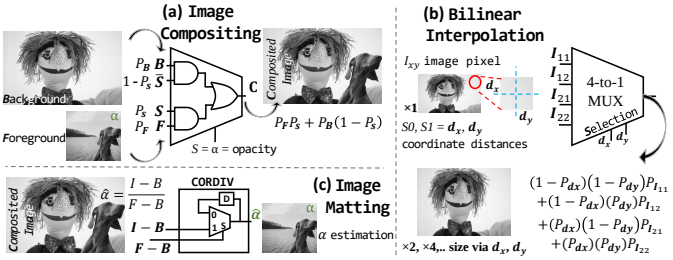


Fig. 3. SC image processing applications: (a) *Image Compositing*, merging background and foreground images with α channel. (b) *Bilinear Interpolation*, up-scaling input images. (c) *Image Matting*, parsing α channel for the foreground object to separate the background.

extracted from [24] and integrated into NVMain [36]. For S-to-B conversion, we consider a single 8-bit ADC from [37] per mat. SL output is prone to failures when deciding on the bitwise operation output due to the intrinsic variability of ReRAM, as described in [38]. We conduct simulations with the VCM-based ReRAM model [39] to determine the distribution of LRS and HRS that leads to the probability of obtaining incorrect outputs in CIM operation. For NVMain simulation, we generate traces for the SBS generation, the SC circuits in Table II, and image processing applications. The derived failure rates are used to simulate fault injections and we report the average results of 1,000 runs. For the CMOS-based SC circuits, we synthesized the designs using the Synopsys Design Compiler with the $45nm$ gate library.

A. Applications

To evaluate our CIM-SC design, we use image processing applications, including (a) image compositing, (b) bilinear interpolation, and (c) image matting.

Image Compositing. In traditional computer vision, two images — background (B) and foreground (F) — are merged using a linear formula that incorporates pixel information from both images, along with an additional channel, α, which represents the opacity of the foreground. This process allows merging two scenes, typically when one of them has a green background. The α-channel defines the object of interest, its refined edges, and the remaining background area in the final composite image. The compositing formula used, is $C = F \times \alpha + B \times (1-\alpha)$, which corresponds to the `MUX` in the SC domain. If the bit-streams of B, F, and α are provided to the first, second, and selection inputs of the `MUX`, respectively, the output becomes the composite image, C. This process is illustrated in Fig. 3(a).

Bilinear Interpolation. Image up-scaling can be achieved through bilinear interpolation involving two linear interpolations along the `x`- and `y`-axes. For any 4-pixel neighboring group, I_{11}, I_{12}, I_{21}, and I_{22}, the input image I can be used to create a larger image by estimating new pixel values between them. The intensity of the new pixel, $I(\mathrm{x},\mathrm{y})$, is calculated using the intensities of the four neighboring pixels and their relative distance (dx, dy) from the new point. The conventional formula is $I(\mathrm{x},\mathrm{y}) = (1-dx)(1-dy)I_{11} + (1-dx)(dy)I_{12} + (dx)(1-dy)I_{21} + (dx)(dy)I_{22}$. This corresponds to a 4-to-1 `MUX` in the SC domain. Here, the bit-streams of the four neighboring pixels serve as inputs to the `MUX`, while the dx and dy bit-streams are fed into the selection ports. The output bit-stream provides the intensity estimate for the new pixel in the up-scaled image. This application is shown in Fig. 3(b).

Image Matting. Image compositing can be reversed to separate the background and foreground images by estimating the α-channel. By rearranging the composite image formula and solving for α, the estimated alpha $\hat{\alpha} = \frac{I-B}{F-B}$ reveals the foreground object and helps refine the edges for a more natural composite image. This estimation is often repeated multiple times with different B and F information, especially when foreground details are missing. Given that this formula involves division operation [40], this work employs the CORDIV design [28], as shown in Fig. 3(c).

B. Performance and Energy Comparison

Table III compares the hardware costs of CMOS-based (✢) and ReRAM-based (✦) SC arithmetic designs. The analysis focuses on the breakdown of SC logic, excluding memory movement costs for CMOS-based designs that further increase their total latency and energy consumption. CMOS implementations employ LFSR and Sobol generators as SNG. Regarding latency, CMOS-based designs are significantly slower than ReRAM-based designs due to the sequential processing of bit-streams. Our ReRAM-based design reduces latency by 38%, compared to CMOS, due to the row-parallel execution of most operations. For the division operation, due to the sequential nature of `flip-flops`, CIM-based CORDIV exhibits higher latency, but it is offset by increased throughput enabled by SIMD parallelism. CORDIV is still compatible with the ADC-based S-to-B conversion, where the SBS is written as resistance values in a column, enabling the ADC to sense the bitline current that represents the number of '1's in the SBS. Other ReRAM-based implementations use the SBS as voltage input, thus requiring a reference column.

If considered in isolation, our ReRAM-based design consumes more energy than CMOS mainly due to multiple read operations ($5N$) for SBS generation and storing SBSs (at least one write operation). In this section, we only report results for the *IMSNG-opt* configuration. For comparison, *IMSNG-naive* requires 395.4 ns and consumes 10.23 nJ per conversion, whereas *IMSNG-opt* completes the same process in 78.2 ns while using only 3.42 nJ.

TABLE III
HARDWARE COST EVALUATION FOR *CMOS-based* (✢) AND *ReRAM-based* (✦) SC TECHNOLOGIES.

CMOS-based Design (✢)			**Total Latency** ❇ (ns)	**Total Energy** (nJ)
Binary→SC ❶	**SC arithmetic operations ❷**	**SC→Binary ❸**		
LFSR + Comparator	Multiplication	$log_2 N$-bit counter	122.88	0.23
	Addition		130.56	0.26
	Subtraction		133.12	0.16
	Division		133.12	0.18
Sobol + Comparator	Multiplication		125.44	0.30
	Addition		130.56	0.30
	Subtraction		133.12	0.12
	Division		130.56	0.14
ReRAM-based Design (✦)				
IMSNG-opt	Multiplication	8-bit ADC [37]	80.8	3.50
	Addition		80.8	3.50
	Subtraction		81.6	3.51
	Division		12544.0	4.48

❇: Total latency=Critical Path Latency × N. Bit-stream Length (N) is 256.

TABLE IV
SSIM (%)/PSNR (dB) COMPARISON FREE OF (✗) OR UNDER (✓) CIM FAULTS ACROSS DIFFERENT BIT-STREAM LENGTHS.

Design	Image Compositing		Bilinear Interpolation		Image Matting	
	✗	✓	✗	✓	✗	✓
✧ [35]	99.9/91.8	82.9/42.4	95.6/39.0	64.4/37.6	99.9/50.3	4.8/-18.2
✦ 32	99.9/23.4	99.9/22.2	82.0/28.5	79.4/28.6	95.3/31.5	88.2/30.0
✦ 64	99.9/26.7	99.9/25.6	87.7/29.5	86.5/29.7	98.7/37.8	93.0/36.7
✦ 128	99.9/28.2	99.9/27.6	91.4/30.2	90.0/29.7	99.4/41.7	94.6/38.5
✦ 256	99.9/32.3	99.9/30.9	93.0/31.1	92.9/31.5	99.7/44.9	96.7/44.5

PSNR: Peak Signal-to-Noise Ratio, SSIM: Structural Similarity. ↑ is better. For the ReRAM-based SC design(✦), different bit-stream lengths (N=32,64,128,256) are used.

Fig. 4 and Fig. 5 compare the SC designs (✢ and ✦) to the binary CIM (✧) (also considering memory transfers). Regarding energy savings, on average, our design reduces energy by 2.8× and 1.15×, compared to the binary CIM and and CMOS designs, respectively.

Only for larger resolutions (N=256), our solution performs poorly compared to CMOS, but these resolutions also considerably increases latency of CMOS-based designs and are typically avoided. In terms of throughput, our design achieves, on average, 2.16× and 1.39× higher throughput compared to the binary CIM and CMOS designs respectively.

The off-chip communication in CMOS-based designs – specifically loading images and storing outputs to the same ReRAM setup – significantly increases total energy consumption. ReRAM-based designs outperform CMOS-based ones for smaller SBSs (32 and 64) across all applications and for all SBS lengths in Bilinear Interpolation. However, for larger SBSs, the cost of writing SBSs outweighs CIM's benefits to the extent that transferring data to the SC logic is more efficient. Nonetheless, implementing SC on general-purpose CIM hardware is a key advantage, as it requires no additional components beyond those common in other CIM designs.

C. Reliability through Stochastic Computing

In digital CIM, a fault is a bit flip, where results invert from the expected value. Table IV presents the quality of our selected applications *with* CIM faults (realistic scenario, ✓) and *without* CIM faults (ideal scenario, ✗). For image compositing and bilinear interpolation, we compare the outputs of *ReRAM-based SC* (✦) and *Binary CIM* (✧) [35] against the SW implementation. For image matting, we compare the blended images obtained using the original α (I in Fig. 3(c)) and the estimated $\hat{\alpha}$. Our design shows an average quality drop of 5% under realistic scenarios. Traditional arithmetic [35] exhibits 47% drop in quality in the presence of faults (as high as 95.2%, in Image Matting), as errors at higher bit positions can lead to more severe and widespread inaccuracies. Among these, image matting relying on integer division is particularly vulnerable to faults, often rendering unacceptable outputs.

Comparing different SBS resolutions shows that some algorithms require only smaller bit-streams (no noticeable drop in accuracy). This aligns well with the performance and energy-efficiency trends presented in Fig. 4, where energy savings increase as bit-stream size decreases. The impact of faults is highly algorithm-dependent. For instance, image compositing and bilinear interpolation, both relying on the MAJ operation,

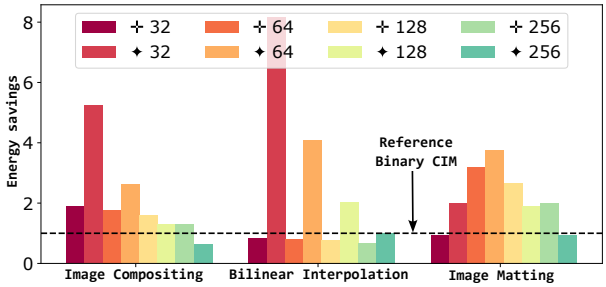


Fig. 4. Normalized energy savings for CMOS (✢) and ReRAM (✦) designs.

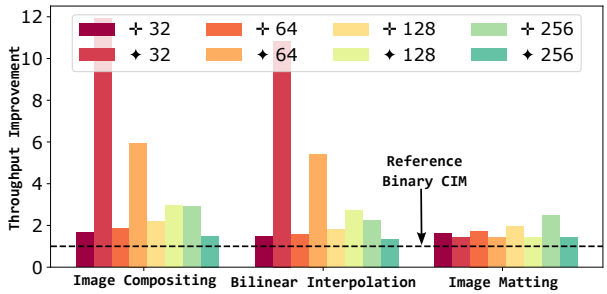


Fig. 5. Normalized throughput for CMOS (✢) and ReRAM (✦) designs.

are more tolerant than CORDIV to CIM faults. This highlights the strength of SC on unreliable CIM, offering more robustness to CIM faults and not requiring dedicated fault protection hardware, making it both energy-efficient and accurate.

Memory protection schemes exist but are costly and traditional error correction codes cannot protect CIM operations (AND, OR, and MAJ). A recent work [41] proposes a parity-based fault-tolerance scheme for NVM-CIM, which adds a significant area overhead for parity storage and syndrome generation, as well as added latency due to critical path dependencies. SC is fault-tolerant, suitable for unreliable devices, and does not add extra overhead to protect CIM operations.

V. Conclusion

We presented a ReRAM-based, in-memory implementation of stochastic computing (SC). Leveraging existing in-ReRAM true RNGs, we produce SBSs, perform stochastic operations, and convert them back to binary – all within the memory array. Compared to the state-of-the-art CMOS-based SC solution, our results across multiple image processing kernels show similar accuracy, 1.39× higher throughput and 1.15× less energy consumption, all without requiring specialized logic for SC.

Acknowledgments

This work is partially funded by National Science Foundation (NSF) under grant No. 2019511, 2339701, the German Research Council (DFG) through the HetCIM project (502388442), the AI competence center ScaDS.AI Dresden/Leipzig (01IS18026A-D), the CRC/TRR 404-Active 3D, and generous gifts from NVIDIA. We thank Stefan Wiefels from Forschungszentrum Jülich for our insightful discussions on the earlier version of this paper.

REFERENCES

[1] C. F. Frasser and et al., "Fully parallel stochastic computing hardware implementation of convolutional neural networks for edge computing applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 12, pp. 10 408–10 418, 2022.

[2] M. W. Daniels, A. Madhavan, P. Talatchian, A. Mizrahi, and M. D. Stiles, "Energy-efficient stochastic computing with superparamagnetic tunnel junctions," *Physical rev. appl.*, vol. 13, no. 3, p. 034016, 2020.

[3] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[4] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE TCAD*, vol. 37, no. 8, 2018.

[5] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 449–462, 2013.

[6] Y.-N. Chang and K. K. Parhi, "Architectures for digital filters using stochastic computing," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 2697–2701.

[7] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing," *ACM Sigplan Notices*, vol. 52, no. 4, pp. 405–418, 2017.

[8] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel sobol sequences," *IEEE TVLSI*, vol. 26, no. 7, pp. 1326–1339, 2018.

[9] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE TVLSI*, vol. 27, no. 12, pp. 2925–2938, 2019.

[10] P. Knag, W. Lu, and Z. Zhang, "A Native Stochastic Computing Architecture Enabled by Memristors," *IEEE Trans. on Nanotechnology*, vol. 13, no. 2, March 2014.

[11] S. Gaba, P. Knag, Z. Zhang, and W. Lu, "Memristive devices for stochastic computing," in *2014 IEEE Intern. Symp. on Circuits and Systems (ISCAS)*, June 2014.

[12] S. Raj, D. Chakraborty, and S. K. Jha, "In-memory flow-based stochastic computing on memristor crossbars using bit-vector stochastic streams," in *2017 IEEE 17th Intern. Conf. on Nanotechnology (IEEE-NANO)*, July 2017, pp. 855–860.

[13] S. Gupta, M. Imani, J. Sim, A. Huang, F. Wu, M. H. Najafi, and T. Rosing, "Scrimp: A general stochastic computing architecture using reram in-memory processing," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1598–1601.

[14] C. Ma, Y. Sun, W. Qian, Z. Meng, R. Yang, and L. Jiang, "Go unary: A novel synapse coding and mapping scheme for reliable reram-based neuromorphic computing," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1432–1437.

[15] Y. Sun, C. Ma, Z. Li, Z. Li, Z. Li, Y. Zhao, J. Jiang, W. Qian, W. Qian, R. Yang, Z. He, L. Jiang, and L. Jiang, "Unary coding and variation-aware optimal mapping scheme for reliable reram-based neuromorphic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[16] C. Lammie, J. K. Eshraghian, W. D. Lu, and M. R. Azghadi, "Memristive stochastic computing for deep learning parameter optimization," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1650–1654, 2021.

[17] M. Riahi Alam, M. Hassan Najafi, and N. TaheriNejad, "Exact stochastic computing multiplication in memristive memory," *IEEE Design Test*, pp. 1–6, 2021.

[18] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[19] V. Gupta, S. Kapur, S. Saurabh, and A. Grover, "Resistive random access memory: a review of device challenges," *IETE Technical Review*, vol. 37, no. 4, pp. 377–390, 2020.

[20] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Scope: A stochastic computing engine for dram-based in-situ accelerator," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 696–709.

[21] K. S. Woo, Y. Wang, J. Kim, Y. Kim, Y. J. Kwon, J. H. Yoon, W. Kim, and C. S. Hwang, "A True Random Number Generator Using Threshold-Switching-Based Memristors in an Efficient Circuit Design," *Advanced Electronic Materials*, vol. 5, no. 2, p. 1800543, 2019.

[22] R. Dittmann, S. Menzel, and R. Waser, "Nanoionic memristive phenomena in metal oxides: the valence change mechanism," *Advances in Physics*, vol. 70, no. 2, pp. 155–349, 2021.

[23] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[24] L. Xie, H. A. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 176–181.

[25] K. Schnieders, P. Bai, Y. Wang, T. Kempen, D. Wouters, V. Rana, R. Waser, S. Menzel, and S. Wiefels, "Exploiting read noise of filamentary VCM ReRAM for efficient TRNG," aug 2024.

[26] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An Architecture for Fault-Tolerant Computation with Stochastic Logic," *Computers, IEEE Trans. on*, vol. 60, no. 1, pp. 93–105, Jan 2011.

[27] M. S. Moghadam, S. Aygun, M. R. Alam, and M. H. Najafi, "P2lsg: Powers-of-2 low-discrepancy sequence generator for stochastic computing," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 38–45.

[28] T.-H. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 116–121.

[29] Y. Zhao, W. Shen, P. Huang, W. Xu, M. Fan, X. Liu, and J. Kang, "A physics-based model of rram probabilistic switching for generating stable and accurate stochastic bit-streams," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 32.4.1–32.4.1.

[30] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. T. Calvino, D. S. Marakkalage *et al.*, "The epfl logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.

[31] C. J. Chevallier, C. H. Siau, S. F. Lim, S. R. Namala, M. Matsuoka, B. L. Bateman, and D. Rinerson, "A 0.13 μm 64mb multi-layered conductive metal-oxide memory," in *2010 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2010, pp. 260–261.

[32] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[33] J. Yu, H. A. Du Nguyen, M. A. Lebdeh, M. Taouil, and S. Hamdioui, "Enhanced scouting logic: A robust memristive logic design scheme," in *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2019, pp. 1–6.

[34] T.-H. Chen and J. P. Hayes, "Equivalence among stochastic logic circuits and its application," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[35] O. Leitersdorf, D. Leitersdorf, J. Gal, M. Dahan, R. Ronen, and S. Kvatinsky, "Aritpim: High-throughput in-memory arithmetic," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 3, pp. 720–735, 2023.

[36] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.

[37] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[38] H. Farzaneh, J. P. De Lima, A. Nezhadi Khelejani, A. A. Khan, M. Mayahinia, M. Tahoori, and J. Castrillon, "Sherlock: Scheduling efficient and reliable bulk bitwise operations in nvms," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[39] S. Wiefels, C. Bengel, N. Kopperberg, K. Zhang, R. Waser, and S. Menzel, "Hrs instability in oxide-based bipolar resistive switching cells," *IEEE Transactions on Electron Devices*, vol. 67, no. 10, pp. 4208–4215, 2020.

[40] M. S. Moghadam, S. Aygun, S. Asadi, and M. H. Najafi, "Low-cost and highly-efficient bit-stream generator for stochastic computing division," *IEEE Transactions on Nanotechnology*, vol. 23, pp. 195–202, 2024.

[41] H. Cılasun, S. Resch, Z. I. Chowdhury, M. Zabihi, Y. Lv, B. Zink, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "On error correction for nonvolatile processing-in-memory," in *International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 678–692.

INSTA: An Ultra-Fast, Differentiable, Statistical Static Timing Analysis Engine for Industrial Physical Design Applications

Yi-Chen Lu, Zhizheng Guo, Kishor Kunal, Rongjian Liang, and Haoxing Ren
NVIDIA Research
{yilu, haoxingr}@nvidia.com

***Abstract*—Prior GPU-accelerated Static Timing Analysis (GPU-STA) works all struggle to find industrial adoption, primarily because they aim to build standalone timing engines that can never emulate the proprietary delay models used in commercial tools. In this paper, we adopt a different philosophy by presenting INSTA, the first-ever differentiable, statistical GPU-STA engine that achieves unprecedented accuracy and scalability by a one-time initialization from *any* reference tool, bringing two transformative capabilities to Physical Design (PD): (1) rapid, high-fidelity timing analysis for incremental netlist update, and (2) gradient-based truly-global timing optimization at scale. Notably, INSTA demonstrates a near-perfect 0.999 correlation with an industry-leading signoff tool on a 15-million-pin design in a commercial $3nm$ node with runtime under 0.1 seconds. Experimental results showcase INSTA's capability through three PD applications: (1) serving as a fast evaluator in an industrial gate sizing flow, achieving 25x faster incremental update_timing runtime with almost no accuracy loss; (2) INSTA-Size, a gradient-based gate sizer that achieves up to 15% better Total Negative Slack (TNS) than the reference signoff engine by sizing 68% fewer amount of cells; and (3) INSTA-Place, a differentiable timing-driven global placer that outperforms the state-of-the-art net-weighting placer by up to 16% in Half-Perimeter Wirelegnth (HPWL) and 59.4% in TNS on the ICCAD'15 benchmark [15].**

I. Introduction

Static Timing Analysis (STA) is the heart of Physical Design (PD) that drives the optimization from synthesis to signoff to achieve precise trade-offs among Power, Performance, and Area (PPA) objectives. With the relentless pursuit of high-performance and low-power designs, STA's role has become even more critical in advanced technology nodes to ensure reliable chip operation under stringent Process, Voltage, and Temperature (PVT) constraints. Despite its pivotal role, commercial STA tools face significant runtime challenge due to the handling of complex constraints and intricate delay calculations required for timing validation. Furthermore, every fundamental netlist transformation, such as gate sizing and buffering, demands iterative STA iterations to evaluate solution quality. While incremental STA methods [12, 26] aim to improve STA runtime by selectively re-propagating affected netlist portions, they often sacrifice precision in optimization, leading to sub-optimal results.

Given the inherently parallel nature of timing propagation and the growing accessibility of GPU programming, recent research has explored GPU-accelerated STA (GPU-STA) engines, utilizing GPU power for both graph-based (GBA) [6, 8, 24, 29] and path-based (PBA) [3–5] timing analysis. Despite their promise, these approaches fail to meet the rigorous accuracy requirements of industrial design flows. They typically rely on simplified delay models, such as the Non-Linear Delay Model (NLDM) for cell delays and the Elmore model for interconnect delays, which lack the acuarcy necessary for advanced technology nodes. In contrast, commercial tools employ proprietary delay models and robustly manage complicated timing constraints. Hence, existing GPU-STA engines demonstrate limited correlation with commercial STA tools. The accuracy gaps are too wide for practical industrial PD applications.

In this paper, we address these limitations by introducing INSTA, a tool-accurate, differentiable, statistical GPU-STA framework tailored for industrial PD flows. INSTA adopts a fundamentally different philosophy from previous works [6, 8, 24, 29] by dividing STA into two distinct stages: delay calculation and timing propagation. Rather than constructing an STA engine from scratch to manage intricate delay computations as prior works, INSTA directly "clones" arc delays from *any* reference tool user-specified and focuses exclusively on ultra-fast, differentiable timing propagation using custom CUDA kernels. This approach combines the strengths of both worlds: (1) unlike prior GPU-STA methods, INSTA achieves high-fidelity correlation with commercial tools, and (2) unlike commercial tools, INSTA enables rapid timing analysis of incremental netlist change and supports gradient-based, truly-global timing optimization subject to various leaf variables (e.g., cell locations, gate sizes).

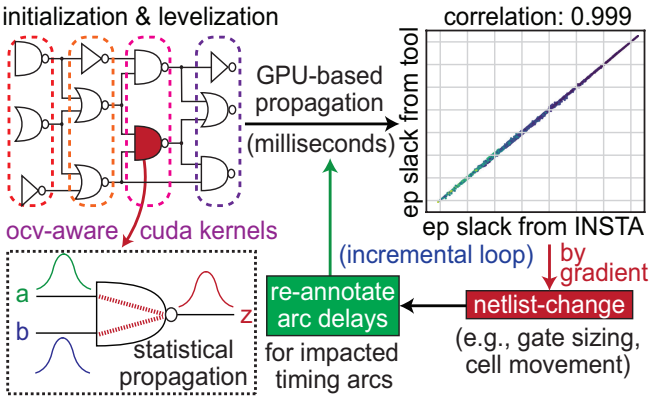


Fig. 1: Overview of INSTA which begins with a one-time initialization from a reference timing engine and performs fast, differentiable, and tool-accurate STA propagation. Notably, INSTA enables gradient computation of global timing metrics (e.g., TNS) with respect to leaf variables (e.g., gate sizes and cell locations) to drive critical PD optimization.

Figure 1 presents an overview of INSTA. Following a one-time initialization from a reference STA engine, INSTA utilizes a forward CUDA kernel for On-Chip Variation (OCV)-aware timing propagation and a backward kernel for gradient backpropagation. Experimental results demonstrate that INSTA achieves a near-perfect correlation (0.9999) in individual endpoint slack values with an industry-leading signoff tool across several real-world high-performance designs in signoff mode. Note that while INSTA is initialized from a reference tool, achieving this level of correlation is far from trivial. INSTA meticulously handles distribution-based timing propagation, timing exceptions, and Common Path Pessimism Removal (CPPR) to manage rise/fall conditions and unateness constraints. Beyond tool-accurate correlation, INSTA supports gradient computations of various leaf optimization variables with respect to global timing metrics including Worst Negative Slack (WNS) and Total Negative Slack (TNS). This capability enables fast and precise identification of timing bottlenecks across cells, nets, stages, etc., facilitating targeted and effective PD optimization. Although in this work, we present applications in gate sizing and global placement, INSTA can be used

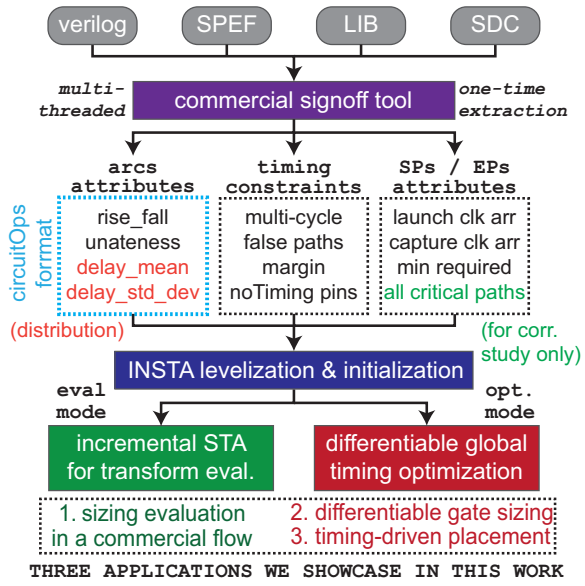


Fig. 2: Overview of INSTA's initialization process from a reference commercial STA tool, along with its evaluation and optimization capabilities.

for many other PD applications such as clock skewing and buffering.

We envision INSTA as a powerful tool to reshape existing industrial PD flows. In this paper, we specifically highlight three novel applications of INSTA. First, we utilize INSTA's fast timing analysis capability within an iterative gate-sizing scheme, demonstrating significant runtime improvements in each sizing iteration with almost no compromise on accuracy or Quality of Results (QoR). Then, we apply INSTA to differentiable gate sizing and differentiable timing-driven placement, illustrating its potential for driving timing optimization in PD tasks. The goal of this work is to demonstrate how INSTA can transform existing CPU-centric PD flows. We believe this paves the way for future works to revisit traditional PD tasks with INSTA.

Our main contributions are as follows:

- We present INSTA, the first-ever differentiable GPU-STA framework that achieves near-perfect endpoint slack correlation to an industry-standard commercial signoff tool[1], in a foundry $3nm$ node with OCV enabled. On our largest design with **15 millions** pins, INSTA performs full-graph timing propagation in less than **0.1 seconds** with 0.999 correlation to the reference signoff tool.
- We develop a GPU-accelerated Top-K statistical arrival propagation CUDA kernel that efficiently manages CPPR, a must-handle timing pessimism in advanced technology nodes.
- We demonstrate INSTA's capability as a fast timing evaluator in a commercial gate sizing flow, where it brings **25x** runtime improvement over reference tool's incremental timing analysis.
- We propose the concept of "timing gradients" to enable truly-global, differentiable timing optimization in traditional PD tasks. We introduce INSTA-Size and INSTA-Place, showing that INSTA-Size outperforms the reference commercial signoff engine by up to **15**% in TNS with sizing **68**% less amount of cells, while INSTA-Place outperforms the state-of-the-art net-weighting-based timing-driven placer [18] by up to **59.4**% in TNS and **16.2**% in Half-Perimeter Wirelength (HPWL).

II. Related Work

A. GPU-Accelerated Timing Propagation

GPU-STA engines have been proposed for graph-based [6, 8, 24, 29] and path-based [3–5] timing analysis, achieving notable speedups on academic benchmarks. However, they rely on over-simplified delay models [19], omit essential industrial requirements such as OCV and CPPR, and lack fidelity to commercial tools. While CPPR-focused methods [7, 11] attempt to address these gaps, their accuracy and runtime are impractical for industrial use. Furthermore, none of these frameworks [3–8, 11, 24, 29] support differentiable timing propagation, limiting their utility for modern PD applications.

[1] The name of the tool is withheld in compliance with our license agreement.

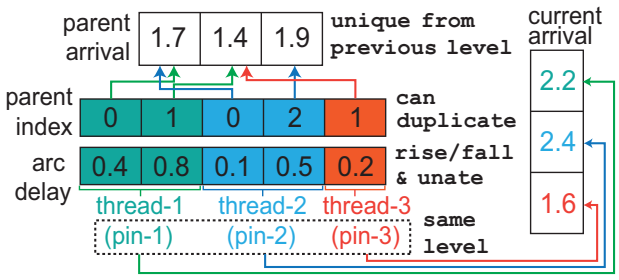


Fig. 3: Illustration (simplified) of arrival time merging in INSTA using GPU, where each pin on the same timing level is mapped to a CUDA thread. Refer to Algorithm 1 for the complete procedure.

B. Differentiable Timing Optimization

Differentiable STA has recently gained traction with the rise of Machine Learning (ML) frameworks like PyTorch [25]. [9] introduces a differentiable placement framework, but its reliance on simplified delay models again limits its usage. Another study [28] leverages ML-based timing predictions [10] to enable gradient computation, but its learning-based nature restricts generalizability, performing 60x slower than commercial tools even on an outdated $130nm$ node. These limitations [9, 28] underscore the need for a framework that combines tool-level accuracy with efficient optimization.

III. Algorithm

A. INSTA's Philosophy and Initialization

The core design principle of INSTA is to achieve tool-accurate, GPU-accelerated timing analysis (GBA), while enabling gradient-based optimization over various leaf variables. However, this alignment is far from trivial. While arc delays can be cloned, replicating the exact timing propagation procedure in a GPU-accelerated manner requires careful handling of OCV, CPPR, and timing constraints. Figure 2 provides an overview of INSTA's initialization process. Inputs include arc attributes in consideration of rise/fall conditions and unateness, timing exceptions (e.g., multi-cycle and false paths), and SP/EP attributes (e.g., clock arrival times, minimum required times). All are extracted using multi-threaded CPU processing via custom TCL scripts into the CircuitOps [17] format. Even on million-gate industrial designs, this extraction only takes around 10 minutes. Following extraction, INSTA constructs and levelizes its timing graph for parallel arrival time propagation, where pins within the same level are processed independently. This levelization process is achieved by topological sort using Graph-Tool [27], which completes in just a few seconds on a million-node graph.

B. OCV Consideration in INSTA: Arrival Times as Distributions

In modern industrial PD flows, STA engines must account for OCV to ensure reliable chip operation under extreme PVT conditions. Although various OCV models are employed across different commercial tools, in this work, we choose to focus on replicating the Parametric OCV (POCV) propagation. To achieve this, INSTA models the arrival time at each pin as a Gaussian distribution that is characterized by a mean and a standard deviation (std). Based on the variational arc delay attributes extracted from the reference tool as shown in Figure 2, our forward CUDA kernel propagates both the mean and standard deviation of these delay distributions at

Algorithm 1 CUDA Kernel for Top-K Statistical **Unique** Arrival Propagation at Output Pins. Considering Rise/Fall and Unateness.

Input: TopK: Number of unique arrivals to store per pin, μ_{arc}: Arc increment means, σ_{arc}: Arc increment standard deviations, $\mathcal{U}$: Arc unateness, $\mathcal{N}_\sigma$: Level of pessimism, N: Number of pins, $outPin_parent_start$: 1D output pins to input pins mapping array.
Output: Top-K arrival distributions per pin subject to **unique** startpoints.

Pre-Kernel Initialization of Final TopK Structures:
1: **Initialize** topK_{rise/fall}_{arrivals, means, stds, SPs} $\in \mathbb{R}^{N\times 2\times K}$
Pre-Kernel Initialization of Temporary Priority Queues:
2: $\{\mathcal{A}, \mu, \sigma, SP\}_{\text{top-K}} \leftarrow \{\} \in \mathbb{R}^{N\times K}$ ▷ arrival, mean, std, startpoint
CUDA Kernel Launch to Parallelize Arrival Merging of Arcs:
3: outPinID $\leftarrow$ blockIdx.x $\cdot$ blockDim.x + threadIdx.x
4: **if** outPinID $\geq N$ **then return** ▷ each maps to a CUDA thread
5: offsetL $\leftarrow$ $outPin_parent_start$[outPinID]
6: offsetR $\leftarrow$ $outPin_parent_start$[outPinID+1]
7: **for each** rf $\in$ {rise,fall}, $k \in \{0,\ldots,\text{TopK}-1\}$ **do**
8: **for each** pID = offsetL **to** offsetR-1 **do** ▷ parent in prev level
9: pRF $=\sim$ rf **if** $\mathcal{U}$[pID] == negative_unate **else** rf
10: pMean $\leftarrow$ topK_means[pID][pRF][k] ▷ from previous level
11: pStd $\leftarrow$ topK_stds[pID][pRF][k]
12: arr_mean $\leftarrow$ pMean$+\mu_{\text{arc}}$[pID][rf]
13: arr_std $\leftarrow \sqrt{\text{pStd}^2+\sigma_{\text{arc}}[\text{pID}][\text{rf}]^2}$
14: arrival $\leftarrow$ arr_mean$+\mathcal{N}_\sigma\cdot$arr_std ▷ $\mathcal{A}_{new}$
15: $\mu_{new} \leftarrow$ arr_mean, $\sigma_{new} \leftarrow$ arr_std
16: $SP_{new} \leftarrow$ topK_SPs[pID][pRF][k]
17: **Update** $\{\mathcal{A}, \mu, \sigma, SP\}_{\text{top-K}}$ at outPinID with **Algorithm 2**
18: **for each** $k \in \{0,\ldots,\text{TopK}-1\}$ **do**
19: topK_arrivals[outPinID][rf][k] $\leftarrow \mathcal{A}_{\text{top-K}}$[outPinID][$k$]
20: topK_means[outPinID][rf][k] $\leftarrow \mu_{\text{top-K}}$[outPinID][$k$]
21: topK_stds[outPinID][rf][k] $\leftarrow \sigma_{\text{top-K}}$[outPinID][$k$]
22: topK_SPs[outPinID][rf][k] $\leftarrow SP_{\text{top-K}}$[outPinID][$k$]
23: **Free** $\{\mathcal{A}, \mu, \sigma, SP\}_{\text{top-K}}$

each timing level, where for an arc $j \rightarrow i$ with its specific variation attributes $arc_mean_{j\rightarrow i}$ and $arc_std_{j\rightarrow i}$, the final arrival time at the sink i is calculated as:

$$arrival_mean_i = arrival_mean_j + arc_mean_{j\rightarrow i}, \quad (1)$$

$$arrival_std_i^2 = arrival_std_j^2 + arc_std_{j\rightarrow i}^2, \quad (2)$$

$$arrival_i = arrival_mean_i + N_\sigma * arrival_std_i, \quad (3)$$

where N_σ controls the level of pessimism. Essentially, the arrival time at the pin i denotes the corner value of its propagated distribution. In the implementation, we set $N_\sigma = 3.0$ in alignment with the reference commercial signoff tool.

C. CPPR Handling in INSTA via Priority Queues

CPPR is essential for accurate timing analysis in industrial design flows, as it mitigates pessimism introduced by clock paths to provide an accurate timing landscape. Prior CPU-based STA methods have attempted to manage CPPR through branch-and-bound techniques for search-space pruning [16, 30], block-based algorithms with alternative delay metrics to reduce pessimism [14], and Lowest Common Ancestor (LCA) path tracing [11]. However, these methods often suffer from substantial runtime overhead or accuracy trade-offs. Recent efforts to utilize GPU parallelism for CPPR path tracing [7] have improved performance but still fall short of industrial requirements.

In INSTA, CPPR is addressed through an efficient GPU-accelerated method that employs Top-K arrival time propagation while ensuring each arrival distribution is mapped to a distinct startpoint. Particularly, priority queues are dynamically maintained at each pin to efficiently track and update the Top-K critical arrival distributions with respect to unique timing startpoints. The key rationale is that clock pessimism arises when multiple timing startpoints converge at a single endpoint along distinct clock paths, resulting in a scenario, where the startpoint

Algorithm 2 GPU-Accelerated Priority Queue Update for Top-K Arrival Propagation with **Unique** Startpoints. (Line 17 of Algorithm 1)

Input: (all from **Algorithm 1**): New arrival time $\mathcal{A}_{\text{new}}$ subject to startpoint $\mathcal{SP}_{\text{new}}$, top-K queues: arrival times $\mathcal{A}_{\text{top-K}}$, means $\mu_{\text{top-K}}$, standard deviations $\sigma_{\text{top-K}}$, startpoints $SP_{\text{top-K}}$.
Output: In-place update of Top-K queues.

At Each GPU Parallel Thread Assigned to an Output Pin:
1: inserted $\leftarrow$ **false**, idx $\leftarrow -1$ ▷ Initialize control flags
Step 1: Check for Startpoint Uniqueness and Update if Found
2: **for each** $(A_j, SP_j) \in$ **zip**($\mathcal{A}_{\text{top-K}}$, $\mathcal{SP}_{\text{top-K}}$) **do**
3: **if** $SP_j = SP_{\text{new}}$ **then**
4: **if** $A_{\text{new}} > A_j$ **then**
5: $\mathcal{A}_{\text{top-K}}[j] \leftarrow A_{\text{new}}$, $\mathcal{SP}_{\text{top-K}}[j] \leftarrow \mathcal{SP}_{\text{new}}$
6: $\mu_{\text{top-K}}[j] \leftarrow \mu_{\text{new}}$, $\sigma_{\text{top-K}}[j] \leftarrow \sigma_{\text{new}}$
7: inserted $\leftarrow$ **true**
8: **break** ▷ Exit once an existing SP_{new} is found
Step 2: Insert New Startpoint if Unique
9: **if not** inserted **then** ▷ Only if SP_{new} is not found
10: **for** $j = 0$ **to** $K-1$ **do**
11: **if** $A_{\text{new}} > A_j$ **then** ▷ Found the first lower value
12: **for** $l = K-1$ **down to** $j+1$ **do** ▷ Shifting in reversal!
13: $\mathcal{A}_{\text{top-K}}[l] \leftarrow \mathcal{A}_{\text{top-K}}[l-1]$
14: $\mathcal{A}_{\text{top-K}}[j] \leftarrow A_{\text{new}}$, $\mathcal{SP}_{\text{top-K}}[j] \leftarrow \mathcal{SP}_{\text{new}}$
15: $\mu_{\text{top-K}}[j] \leftarrow \mu_{\text{new}}$, $\sigma_{\text{top-K}}[j] \leftarrow \sigma_{\text{new}}$
16: inserted $\leftarrow$ **true**
17: **break**

contributing to the maximum arrival time at an endpoint, differs from the startpoint introducing the minimum required time, which causes a shift in the endpoint's slack. Hence, with the Top-K arrival times that are mapped to unique startpoints and by initializing endpoint required times from a commercial reference tool that are also associated with respective startpoints (as illustrated in Figure 2), INSTA ensures that the correct arrival time is used to compute endpoints' slack values for tool-accurate timing analysis.

D. GPU Kernel Implementation of Top-K Arrival Propagation

Figure 3 shows how INSTA uses GPU parallelism to perform arrival time propagation in large-scale timing analysis. At each timing level, an index array in GPU shared memory maps each thread, representing a pin at the current level, to its parent pins. This mapping enables accurate merging of arrival distributions across multiple incoming arcs while handling rise/fall conditions and unateness constraints. Algorithm 1 provides the detailed steps for this process, focusing on output pins where multiple arcs converge. Note that for input pins, GPU kernels are not required, as each input pin is connected to a single parent in modern digital designs. Hence, INSTA employs a CPU-based vectorized approach, which avoids the overhead of data transfer to GPU while being computationally efficient.

The update of priority queues for CPPR handling is detailed in Line 17 of Algorithm 1, with the specific procedures outlined in Algorithm 2, which ensures that each Top-K entry corresponds to a unique startpoint while dynamically maintaining the Top-K arrival times. The update procedure works as follow. If the new incoming startpoint $\mathcal{SP}_{\text{new}}$ already exists in the queue, its arrival time $\mathcal{A}_{\text{new}}$ is compared with the existing value, and the entry is updated if $\mathcal{A}_{\text{new}}$ is larger. If $\mathcal{SP}_{\text{new}}$ is not found in the queue, the algorithm checks whether $\mathcal{A}_{\text{new}}$ exceeds the smallest arrival time in the current list. If so, the new values are inserted in the appropriate position to maintain descending order, with entries shifted as needed.

E. INSTA Complexity Analysis: Why not Heaps for Priority Queues?

In conventional priority queue implementations, heaps are commonly used due to their $O(\log K)$ complexity for insertions and deletions. However, implementing a heap on GPUs incurs substantial overhead because maintaining a binary tree structure requires frequent reorder-

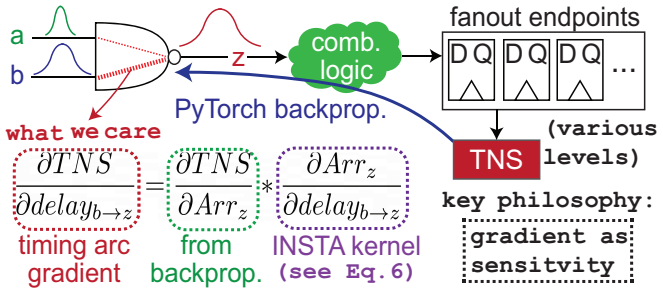


Fig. 4: Illustration of timing gradients in INSTA. For either cell arc or net arc, we can compute the "timing gradient" that quantifies its contribution to global timing metrics such as WNS and TNS.

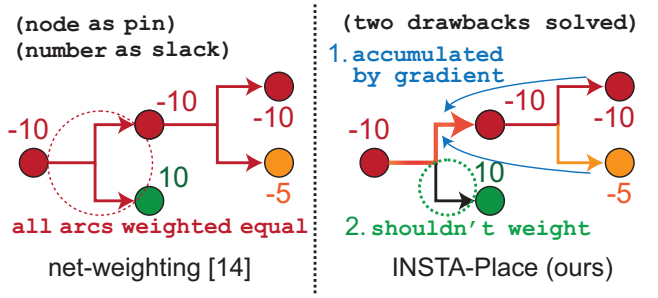


Fig. 5: Illustration of arc-based gradient for timing-driven placement in INSTA-Place that solves the two key drawbacks of the state-of-the-art net-weighting-based approach [18].

ing and re-balancing of elements. These operations are computationally intensive and poorly suited to the highly parallel architecture of GPUs, where hundreds thousands of threads operate simultaneously. In INSTA, we use a fixed-size list to manage the Top-K elements, avoiding the overhead of heap-based operations. As shown in the two algorithms, each CUDA thread operates independently on its own lists, performing $O(K^2)$ comparisons and shifts to maintain the sorted order. Hence, with the GPU ensuring parallel processing across threads in $O(1)$ per level, the overall time complexity of INSTA's timing propagation is $O(K^2 \cdot L)$, where K denotes Top-K and L is the number of timing levels.

F. Enabling Differentiable Timing Propagation

In Algorithm 1 and Algorithm 2, the "greater than" operation identifies the maximum arrival time at a pin from multiple incoming paths. While effective for evaluation, it introduces non-differentiability, making it unsuitable for optimization. The limitation arises from its exclusivity, propagating only the arrival distribution of the most critical path and blocking gradient flow from other inputs. This approach is particularly problematic when multiple input paths provide near-critical arrival times, as it disregards opportunities to optimize other sub-critical paths to improve overall timing. To overcome this, we adopt the numerically stable Log-Sum-Exponential (LSE) operator, a differentiable approximation of the maximum function. The LSE operator ensures smoother gradient distribution across all inputs, enabling more comprehensive optimization. It is defined as:

$$\text{LSE}(\{\mathcal{A}_i\}_{i=1}^n) = M + \tau \cdot \log\left(\sum_{i=1}^{n} \exp\left(\frac{\mathcal{A}_i - M}{\tau}\right)\right), \quad (4)$$

where $\{\mathcal{A}_i\}$ denotes the set of arrival times at a given pin, $M = \max(\{\mathcal{A}_i\}_{i=1}^n)$ ensures numerical stability, and $\tau > 0$ is a temperature parameter controlling the degree of smoothness. As $\tau \to 0$, the LSE operator converges to the standard maximum operator as:

$$\lim_{\tau \to 0} \text{LSE}(\{\mathcal{A}_i\}_{i=1}^n) = \max(\{\mathcal{A}_i\}_{i=1}^n). \quad (5)$$

With Equation 4, the gradient of LSE can be computed in INSTA's backward kernel with respect to each arrival time $\mathcal{A}_i$ as:

$$\frac{\partial \text{LSE}(\{\mathcal{A}_i\}_{i=1}^n)}{\partial \mathcal{A}_i} = \frac{\exp\left(\frac{\mathcal{A}_i - M}{\tau}\right)}{\sum_{j=1}^{n} \exp\left(\frac{\mathcal{A}_j - M}{\tau}\right)}, \quad (6)$$

which effectively assigns a softmax-like weight to each path. The term $\exp\left(\frac{\mathcal{A}_i - M}{\tau}\right)$ ensures that paths with larger arrival times receive higher contributions, while smaller paths contribute proportionally less. Notably, this formulation ensures full differentiability, allowing gradient descent to consider full timing graph during optimization.

G. Timing Gradient: Key for Truly-Global Timing Optimization

For the first time, in the realm of PD, INSTA introduces the concept of "timing gradients" to enable arc-based, truly-global timing optimization for industrial-grade PD applications. The key idea is that the timing gradient of each arc, accounting for rise/fall and unateness conditions, precisely quantifies its contribution to global timing metrics such as WNS and TNS, which enables targeted and fine-grained PD optimization. If we draw an analogy between INSTA and modern ML models, each pin in the timing graph corresponds to a neuron, and the pin-to-pin arc delays can be thought as weights. Similar to how forward propagation in a ML model that computes a differentiable loss function with respect to the weights, INSTA computes timing metrics in a fully differentiable manner.

Figure 4 illustrates the computation of timing gradients in INSTA. Leveraging PyTorch's C++/CUDA extension [25] (similar to DREAMPlace [20]), INSTA integrates PyTorch's auto-differentiation capabilities to perform efficient gradient backpropagation. This design choice allows INSTA to delegate the complicated global gradient flow to the PyTorch framework while focusing solely on gradient computations with a custom backward CUDA kernel at each timing level. As illustrated in the purple box of Figure 4, INSTA's backward kernel computes the gradient of merged arrival times at a destination pin to incremental changes in arc delays (e.g., $\frac{\partial Arr_z}{\partial delay_{b\to z}}$), following the formulation in Equation 6. The timing gradient of each arc is subsequently derived by combining this kernel's output with the backpropagated gradient contributions from the previous fanout level. In the experiments, we demonstrate how to leverage "timing gradient" to drive critical PD optimization including gate sizing and placement.

H. INSTA-Size: Pinpoint Gate Sizing with Timing Gradients

As INSTA computes the "timing gradient" of each arc in a GPU-accelerated manner, it enables **instantaneous** identification of timing-critical cells or nets within a design. This capability allows us to quickly pinpoint which elements contribute most significantly to global timing metrics such as TNS and prioritize them 'for timing improvement. Leveraging this concept, we introduce INSTA-Size, a fast, gradient-based gate sizing framework designed to optimize timing using the gradients calculated by INSTA. The core idea is that following the one-time initialization, a backward pass on the TNS metric computed by INSTA yields the timing gradient of each "stage" (i.e., the gradient sum of a cell arc and its driving net arc), which quantifies its contribution to the overall TNS. Stages with gradients above a pre-defined threshold are prioritized by magnitude. Starting with the most critical stage, a TCL command `EstimateEco` is used to determine the library cell that provides the most delay improvement for that stage, which is then committed and rolled back if TNS degrades. Note that `EstimateEco` is a highly parallelizable command, which computes local delay change estimates of gate sizing for millions of arcs within seconds in the reference tool. A recent work [22] has shown that the accuracy of `EstimateEco` is good enough to drive commercial sizing optimization. Nonetheless, as `EstimateEco` operates under the assumption that the neighboring

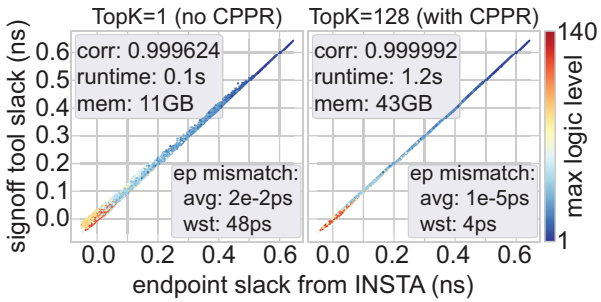


Fig. 6: INSTA vs. commercial signoff tool endpoint slack correlation on block-1 with 4M cells and 15M pins. Top-K=1 and Top-K=128 propagation are compared, showing the trade-off of CPPR consideration.

cells remain unchanged for estimation, INSTA-Size blocks the 3-hop neighborhood of a committed stage from further consideration to mitigate interference, which aligns with the strategies used in [22, 23].

I. INSTA-Place: INSTA for Timing-Driven Placement

Now, we introduce INSTA-Place, which is a timing-driven placer that leverages timing gradients from INSTA for global placement optimization. By integrating with DREAMPlace [20] and OpenTimer [13], INSTA-Place addresses the fundamental limitations in the state-of-the-art net-weighting approach [18], as shown in Figure 5, via an arc-specific weighting strategy guided by timing gradients. Particularly, the two critical issues that INSTA-Place solves are:

- **Locality of slack-based adjustment**: Current methods assign net weights primarily based on the most critical sink, leading to near-uniform weighting for arcs along the same worst critical path. However, instead of relying on "slack values", the weighting strategy should reflect each arc's contribution (i.e., gradient) to global timing metrics such as TNS.
- **Equal weighting for all arcs in a net**: Existing methods assign the same weight to all arcs in a net, which is inherently suboptimal. Often, only a subset of sinks in a net are timing-critical, while the others are not. Blindly forcing all pins closer together degrades overall placement quality.

INSTA-Place elegantly resolves these limitations by using an arc-based, "gradient as sensitivity" approach by weighting pin-to-pin Manhattan distances by their respective timing gradients from INSTA. Specifically, at each placement iteration, given the current pin locations $\mathbf{P} = \{(x_i, y_i)\}_{i=1}^{N}$, the critical timing arc indices $\mathbf{I} = \{(f_k, t_k)\}_{k=1}^{M}$ (where f_k and t_k denote the indices of the driver and sink of the k-th arc), and the associated timing gradients $\mathbf{G} = \{g_k\}_{k=1}^{M}$ from INSTA, we define the timing objective as:

$$L_{\text{timing}} = \lambda_{RC} \cdot \sum_{k=1}^{M} \left(|x_{f_k} - x_{t_k}| + |y_{f_k} - y_{t_k}| \right) \cdot g_k, \tag{7}$$

where λ_{RC} denotes the global time constant scaling that reflects the RC delay per wirelength. To integrate timing-awareness into the global placement process, we combine L_{timing} with the default wirelength L_{WL} and density L_{DEN} objectives as:

$$L = \underbrace{L_{\text{WL}} + \lambda_1 L_{\text{den}}}_{\text{default objective}} + \underbrace{\lambda_2 L_{\text{timing}}}_{\text{INSTA-Place}}, \text{ s.t. } \lambda_2 = \frac{\|\nabla (L_{\text{WL}} + \lambda_1 L_{\text{den}})\|}{\|\nabla L_{\text{timing}}\|} \tag{8}$$

where λ_1 is the default density scaling factor, and λ_2 dynamically adjusts the contribution of L_{timing} relative to the default wirelength and density objectives by aligning the Euclidean norm $\|\cdot\|$ of their gradients. For computational efficiency, INSTA-Place refreshes the timing graph with OpenTimer [13] every 15 iterations as the practice in [18], while reusing last-computed timing gradients for the remaining 14 iterations. This approach ensures accurate timing analysis without incurring excessive computational overhead.

TABLE I: Timing correlation study between INSTA and an industry-leading signoff tool on 5 industrial designs in a commercial $3nm$ node. UT refers to the `update_timing` runtime of the tool with 32 threads.

designs (TopK=32) # cells, # pins, UT(min)	ep slack corr. (top 5 digits)	runtime (second)	memory (GB)	ep mismatch (avg, wst) **ps**
block-1 (4M, 15M, 68)	0.99994	0.39	21.13	(7e-4, 17)
block-2 (2M, 6M, 32)	0.99999	0.35	5.98	(1e-4, 3)
block-3 (3M, 9M, 39)	0.99992	0.37	11.94	(1e-3 , 9)
block-4 (2M, 7M, 29)	0.99996	0.35	9.47	(2e-4, 6)
block-5 (2M, 6M, 33)	0.99999	0.33	5.81	(1e-4, 5)

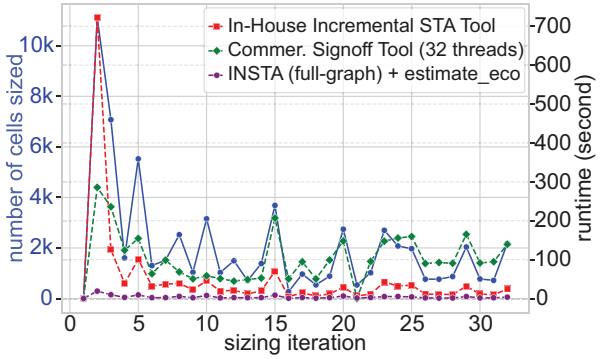


Fig. 7: Incremental STA runtime comparison (same changelist) on block-2 among an in-house CPU-based STA engine, a commercial signoff tool, and INSTA with `EstimateEco` for arc delay re-annotation.

IV. Experimental Results

In this section, we first present a detailed correlation analysis between INSTA and an industry-leading signoff tool with 5 real-world million-gate industrial designs in a commercial $3nm$ technology node. Then, we demonstrate INSTA's capabilities across three PD applications as illustrated in Figure 2. INSTA is implemented using PyTorch [25] 2.1.2 and CUDA 12.1. All experiments are conducted on a Linux system equipped with a single NVIDIA A100 GPU (96GB memory) and an AMD EPYC 7742 64-core processor with 2TB RAM.

A. Correlation Study: INSTA vs. Commercial Signoff Tool

In this experiment, we validate INSTA's timing correlation against an industry-leading commercial signoff tool. Figure 6 shows the endpoint slack correlation results as scatter plots on block-1 (4M cells, 15M pins), with the left plot achieved without CPPR handling (Top-K=1) and the right plot with consideration of CPPR (Top-K=128). Each dot in the figure represents an endpoint and is colored by its maximum level, highlighting INSTA's accuracy across varying timing depths. With CPPR handling, correlation is improved with expected trade-offs in runtime and memory. However, even without CPPR (Top-K=1), INSTA shows exceptional accuracy, with an average absolute mismatch per endpoint of only **0.02ps**, indicating near-perfect alignment with the 45-degree line. Table I presents detailed timing correlation results across 5 industrial designs with a fixed Top-K=32. Notably, the tool's `update_timing` runtimes are listed only for reference rather for direct comparison with INSTA, as INSTA focuses solely on timing propagation using arc delays initialized from the tool to enable incremental timing evaluation and differentiable timing optimization. However, it is worth to note that since GPU parallelism amortizes the computation at each timing level to $O(1)$, INSTA's runtime scales with the number of timing levels under a fixed Top-K rather than the total number of pins or cells.

B. Application-1: Timing Evaluation in a Commercial Sizing Flow

Now, we evaluate INSTA's capability as a timing evaluator within a commercial gate sizing flow for timing-constrained power optimization in a $3nm$ node, where incremental STA is used after each iteration to ensure timing consistency. In this experiment, we

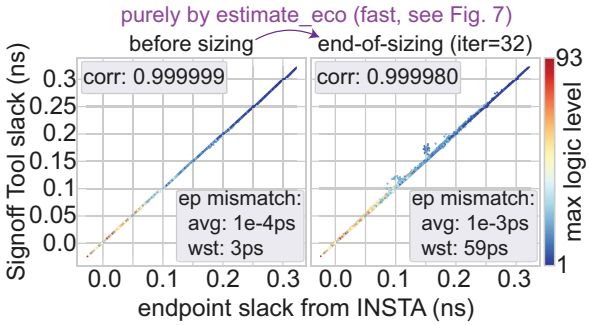


Fig. 8: INSTA correlation impact with `EstimateEco` re-annotation before and after a commercial gate sizing flow for power optimization. The correlation is still high enough to drive the optimization.

TABLE II: Gate sizing for timing optimization results between INSTA-Size and a commercial signoff tool on IWLS benchmark [1]. bRT denotes INSTA's runtime of identifying critical stages via the backward kernel.

design (# pins) (bRT, RT_{pt})	method	WNS (ps)	TNS (ps)	# vio. eps	# cells sized
aes (24k) (0.02s, 33s)	initial state	-27.59	-1115.81	149	–
	signoff tool	-27.59	-1206.12	152	182
	INSTA-Size	-27.59	-1025.15	137	113 (-38%)
cipher_top (50k) (0.02s, 42s)	initial state	-69.24	-11232.28	317	–
	signoff tool	-69.00	-11187.22	314	431
	INSTA-Size	-68.07	-10924.20	311	254 (-41%)
des (11k) (0.03s, 14s)	initial state	-114.61	-5546.59	113	–
	signoff tool	-114.50	-5498.41	112	186
	INSTA-Size	-113.54	-5360.59	110	121 (-35%)
mc_top (25k) (0.02s, 22s)	initial state	-203.01	-77653.10	776	–
	signoff tool	-203.00	-77694.79	774	156
	INSTA-Size	-203.01	-75307.32	776	49 (-68%)

benchmark INSTA's runtime and correlation against two other STA engines in incremental mode: an in-house, highly-optimized STA tool, and a commercial signoff tool with 32 threads. Figure 7 shows the incremental STA runtime comparison at each sizing iteration over the exact same changelist across these three tools. Particularly, it is important to note that the runtime of INSTA includes both timing propagation time and the arc delay re-annotation time from the commercial tool's `EstimateEco`. Figure 8 further evaluates INSTA's correlation before and after the complete gate sizing flow against the reference signoff tool. While `EstimateEco` introducing minor inaccuracies which can be observed in the right figure, INSTA maintains extremely high correlation throughout the entire optimization process for the vast majority of endpoints, delivering a **14x** speed-up over the highly-optimized in-house STA engine, and a **25x** runtime improvement over the reference commercial tool. We want to emphasize that in practice, any significant accuracy concerns can be addressed by re-synchronizing INSTA with tool-calculated arc delays, as shown in Figure 2, which takes only 10 minutes for million-gate industrial designs. Note that in this experiment, we do not perform this re-synchronization for benchmarking purpose.

C. Application-2: INSTA-Size for Timing Optimization

In this experiment, we assess INSTA-Size's capability in gate sizing for timing optimization using the "timing gradients" as described in Section III-H. A head-to-head comparison between INSTA-Size and the commercial signoff tool is conducted on the IWLS benchmark suite [1] in the ASAP $7nm$[2] node, with $\tau = 0.01$ set in Equation4 for INSTA-Size. Table III-H presents the optimization results, demonstrating that INSTA-Size achieves up to 15% improvement in TNS compared to the commercial tool's default timing optimization engine, while requiring significantly fewer cell modifications across all benchmarks (**42%** less on average). We attribute this significant success to the effectiveness of INSTA's timing gradients, which offer a precise and instant identification of the most critical stages and their corresponding sensitivities to TNS (i.e., gradients as sensitivities).

TABLE III: Timing-driven placement results **after legalization** between INSTA-Place and the state-of-the-art net-weighting-based approach [18] on ICCAD 2015 benchmarks [15]. The unit for TNS is $10^5 ps$. DP denotes DREAMPlace. Refer to Figure 5 for detailed runtime comparison.

ICCAD'15 benchmark	DP [20]		DP 4.0 [18]		INSTA-Place (ours)	
	HPWL	TNS	HPWL	TNS	HPWL	TNS
Superblue1	410.4	-312.5	481.6	-85.3	443.1(-8.0%)	-34.6(-59.4%)
Superblue3	469.3	-70.3	482.7	-50.4	472.2(-2.2%)	-40.4(-19.8%)
Superblue4	317.3	-191.1	343.7	-144.0	333.9(-2.9%)	-114.5(-21%)
Superblue5	487.9	-211.6	533.5	-102.1	478.5(-10.3%)	-93.7(-8.2%)
Superblue7	600.2	-167.3	604.1	-64.7	593.0(-1.8%)	-57.6(-10.9%)
Superblue10	916.0	-756.0	1088.3	-671.3	911.9(-16.2%)	-628.2(-6.4%)
Superblue16	424.4	-250.1	460.2	-71.4	458.7(-0.3%)	-37.6(-47.3%)
Superblue18	237.1	-92.2	248.6	-47.5	242.6(-2.4%)	-35.8(-24.6%)
					avg. -5.5%	avg. -24.7%

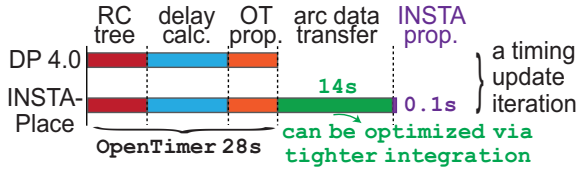


Fig. 9: Detailed runtime comparison between [18] and INSTA-Place on Superblue10 (largest benchmark, 5.6M pins) in a timing update iteration.

D. Application-3: INSTA-Place for Timing-Driven Global Placement

In this experiment, we evaluate INSTA-Place against DREAMPlace 4.0 [18], the state-of-the-art open-source net-weighting-based timing-driven placer, using the well-established ICCAD 2015 benchmarks [15]. The methodology of INSTA-Place is detailed in Section III-I, with post-legalization results (by ABCDPlace [21]) summarized in Table III with λ_{RC} set to around 0.001. It is shown that INSTA-Place achieves up to 59.4% TNS improvement and 16.2% wirelength reduction compared to [18]. Figure 9 details the runtime comparison between INSTA-Place and [18] on Superblue10 over a timing update iteration. While INSTA-Place incurs a 50% runtime overhead due to the data transfer between OpenTimer [13] and INSTA, this overhead can be minimized through tighter integration (e.g., shared data structures). Notably, runtime is not the focus of this experiment. What we aim to demonstrate is that by addressing the fundamental shortcomings of [18] as highlighted in Figure 5 through the arc-based weighting strategy, INSTA-Place can significantly advance the timing quality of global placement. Finally, while prior work [9] also attempts differentiable timing-driven placement by approximating gradient propagation over non-differentiable variables such as Steiner points, this approach is error-prone and poorly correlates with reference STA tools. In contrast, INSTA-Place employs arc-based timing gradients for distance-based weighting, providing a more elegant and robust approach. On benchmarks like Superblue1 and Superblue16, INSTA-Place achieves over 50% TNS improvement compared to [9], while achieving parity results on other benchmarks.

V. Conclusion

We have proposed INSTA, a GPU-accelerated, differentiable STA engine tailored for industrial PD flows. INSTA achieves near-perfect correlation with an industry-leading commercial signoff tool while delivering exceptional computational efficiency. By utilizing timing gradients, INSTA enables highly effective optimization in critical PD tasks, including gate sizing and timing-driven placement, achieving significant improvements in QoR and resource utilization. We believe INSTA shall pave the way for transforming existing CPU-centric PD flows with high-fidelity GPU-STA.

REFERENCES

[1] C. Albrecht. Iwls 2005 benchmarks. In International Workshop for Logic Synthesis (IWLS), volume 9, 2005.

[2] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric. Asap7: A 7-nm finfet predictive process design kit. Microelectronics Journal, 53:105–115, 2016.

[3] G. Guo, T.-W. Huang, Y. Lin, Z. Guo, S. Yellapragada, and M. D. Wong. A gpu-accelerated framework for path-based timing analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42(11):4219–4232, 2023.

[4] G. Guo, T.-W. Huang, Y. Lin, and M. Wong. Gpu-accelerated critical path generation with path constraints. In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9. IEEE, 2021.

[5] G. Guo, T.-W. Huang, Y. Lin, and M. Wong. Gpu-accelerated path-based timing analysis. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 721–726. IEEE, 2021.

[6] Z. Guo, T.-W. Huang, and Y. Lin. Gpu-accelerated static timing analysis. In Proceedings of the 39th international conference on computer-aided design, pages 1–9, 2020.

[7] Z. Guo, T.-W. Huang, and Y. Lin. Heterocppr: Accelerating common path pessimism removal with heterogeneous cpu-gpu parallelism. In 2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). ACM, 2021.

[8] Z. Guo, T.-W. Huang, and Y. Lin. Accelerating static timing analysis using cpu–gpu heterogeneous parallelism. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42(12):4973–4984, 2023.

[9] Z. Guo and Y. Lin. Differentiable-timing-driven global placement. In Proceedings of the 59th ACM/IEEE Design Automation Conference, pages 1315–1320, 2022.

[10] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In Proceedings of the 59th ACM/IEEE Design Automation Conference, pages 1207–1212, 2022.

[11] Z. Guo, M. Yang, T.-W. Huang, and Y. Lin. A provably good and practically efficient algorithm for common path pessimism removal in large designs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(10):3466–3478, 2021.

[12] T.-W. Huang, G. Guo, C.-X. Lin, and M. D. Wong. Opentimer v2: A new parallel incremental timing analysis engine. IEEE transactions on computer-aided design of integrated circuits and systems, 40(4):776–789, 2020.

[13] T.-W. Huang and M. D. Wong. Opentimer: A high-performance timing analysis tool. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 895–902. IEEE, 2015.

[14] B. Jin, G. Luo, and W. Zhang. A fast and accurate approach for common path pessimism removal in static timing analysis. In 2016 IEEE International Symposium on Circuits and Systems (ISCAS), pages 2623–2626. IEEE, 2016.

[15] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 921–926. IEEE, 2015.

[16] P.-Y. Lee, I. H.-R. Jiang, C.-R. Li, W.-L. Chiu, and Y.-M. Yang. itimerc 2.0: Fast incremental timing and cppr analysis. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 890–894. IEEE, 2015.

[17] R. Liang, A. Agnesina, G. Pradipta, V. A. Chhabria, and H. Ren. Circuitops: An ml infrastructure enabling generative ai for vlsi circuit optimization. In 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), pages 1–6. IEEE, 2023.

[18] P. Liao, S. Liu, Z. Chen, W. Lv, Y. Lin, and B. Yu. Dreamplace 4.0: Timing-driven global placement with momentum-based net weighting. In 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 939–944. IEEE, 2022.

[19] S. Lin, G. Guo, T.-W. Huang, W. Sheng, E. F. Young, and M. D. Wong. Gcs-timer: Gpu-accelerated current source model based static timing analysis. In Proceedings of the 61th ACM/IEEE Design Automation Conference, 2024.

[20] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(4):748–761, 2020.

[21] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan. Abcdplace: Accelerated batch-based concurrent detailed placement on multithreaded cpus and gpus. IEEE transactions on computer-aided design of integrated circuits and systems, 39(12):5083–5096, 2020.

[22] Y.-C. Lu, K. Kunal, G. Pradipta, R. Liang, R. Gandikota, and H. Ren. Lego-size: Llm-enhanced gpu-optimized signoff-accurate differentiable vlsi gate sizing in advanced nodes. In Proceedings of the 2025 International Symposium on Physical Design. ACM, 2025.

[23] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 733–738. IEEE, 2021.

[24] K. E. Murray and V. Betz. Tatum: Parallel timing analysis for faster design cycles and improved optimization. In 2018 International Conference on Field-Programmable Technology (FPT), pages 110–117. IEEE, 2018.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 2019.

[26] C. Peddawad, A. Goel, B. Dheeraj, and N. Chandrachoodan. iitrace: A memory efficient engine for fast incremental timing analysis and clock pessimism removal. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 903–909. IEEE, 2015.

[27] T. P. Peixoto. The graph-tool python library. figshare, 2014.

[28] P. Pham and J. Chung. Agd: A learning-based optimization framework for eda and its application to gate sizing. In 2023 60th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2023.

[29] H. H.-W. Wang, L. Y.-Z. Lin, R. H.-M. Huang, and C. H.-P. Wen. Casta: Cuda-accelerated static timing analysis for vlsi designs. In 2014 43rd International Conference on Parallel Processing, pages 192–200. IEEE, 2014.

[30] Y.-M. Yang, Y.-W. Chang, and I. H.-R. Jiang. itimerc: Common path pessimism removal using effective reduction methods. In 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 600–605. IEEE, 2014.

NVR: Vector Runahead on NPUs for Sparse Memory Access

Hui Wang[1*], Zhengpeng Zhao[4*], Jing Wang[1], Yushu Du[1], Yuan Cheng[2,3], Bing Guo[1], He Xiao[5],
Chenhao Ma[1], Xiaomeng Han[1], Dean You[1], Jiapeng Guan[6], Ran Wei[6] Dawei Yang[2✉], Zhe Jiang[1✉]
[1]*National Center of Technology Innovation for EDA, School of Integrated Circuits, Southeast University* [2]*Houmo AI*
[3]*Nanjing University* [4]*Huazhong University of Science and Technology* [5]*Harbin Institute of Technology* [6]*Dalian University of Technology*
whmio0115@seu.edu.cn, u202114911@hust.edu.cn, yuancheng@smail.nju.edu.cn, dawei.yang@houmo.ai, zhejiang.uk@gmail.com

***Abstract*—Deep Neural Networks are increasingly leveraging sparsity to reduce the scaling up of model parameter size. However, reducing wall-clock time through sparsity and pruning remains challenging due to irregular memory access patterns, leading to frequent cache misses. In this paper, we present NPU Vector Runahead (NVR), a prefetching mechanism tailored for NPUs to address cache miss problems in sparse DNN workloads. Rather than optimising memory patterns with high overhead and poor portability, NVR adapts runahead execution to the unique architecture of NPUs. NVR provides a general micro-architectural solution for sparse DNN workloads without requiring compiler or algorithmic support, operating as a decoupled, speculative, lightweight hardware sub-thread alongside the NPU, with minimal hardware overhead (under 5%). NVR achieves an average 90% reduction in cache misses compared to SOTA prefetching in general-purpose processors, delivering 4x average speedup on sparse workloads versus NPUs without prefetching. Moreover, we investigate the advantages of incorporating a small cache (16KB) into the NPU combined with NVR. Our evaluation shows that expanding this modest cache delivers 5x higher performance benefits than increasing the L2 cache size by the same amount.**

I. Introduction

Emerging Deep Neural Networks (DNNs), particularly Large Language Models (LLMs), often scale to hundreds of billions of parameters [1], increasingly consuming more storage, memory bandwidth, and computational resources. Fortunately, these workloads are typically over-parameterised [2], where up to 90% of parameters in prevalent models can be pruned while maintaining comparable performance [3]. This redundancy presents an opportunity to leverage sparsity to reduce such intensive resource demands.

Theoretically, more fine-grained sparsity patterns yield higher acceleration by skipping more zero-valued elements. However, as shown in Fig. 1 ⓐ, these patterns introduce unstructured characteristics when directly skipping zero values randomises vector access order, leading to irregular memory access patterns. These irregular patterns frequently result in cache misses, which not only waste main memory bandwidth but also degrade NPU performance. Taking sparse KVCache in LLM workloads [4] as an example, Fig. 1 ⓑ illustrates that despite a 16x parameter reduction, frequent cache misses lead to extensive main memory interactions. This results in only a 5x actual speedup, substantially below theoretical expectations. Cache misses thus emerge as a key barrier to sparse workloads.

Existing Work. Recent research has focused on mitigating the side effects of fine-grained sparsity. Farshchi et al. [5] propose a bitmask sparse data format in NVDLA, while Eyeriss [6] design a run-length encoding for non-zero elements. Albericio et al. [7] introduce additional mapping algorithms, regularising memory access patterns through label cache mechanisms. Meanwhile, Liu [8] et al. explored sparse domain compute units, developing a butterfly-based architecture for efficient sparse data structure processing. Whilst most of these approaches attempt to regularise memory access patterns to avoid cache misses, they often introduce substantial area and control logic overhead that outweighs their benefits; therefore, the practical applicability of these algorithm-specific are significantly restricted.

Contributions: Different from all previous works, we tackle the sparsity challenge from a different perspective. Instead of regularising irregular memory access patterns, our approach embraces the inherent characteristics of sparse computations. We leverage prefetching techniques directly to mitigate the impact of cache misses, rather than attempting to eliminate the irregularity itself, which incurs high overhead and lacks generality. We propose a vector runahead mechanism for NPU (NVR), drawing inspiration from the runahead execution model [9], [10]. NVR integrates a lightweight prefetcher decoupled with CPU and NPU, achieving significant cache miss reduction through speculative execution utilising idle NPU units. NVR is a micro-architectural solution requiring no compiler or algorithmic support, remains orthogonal to prior optimisation methods.

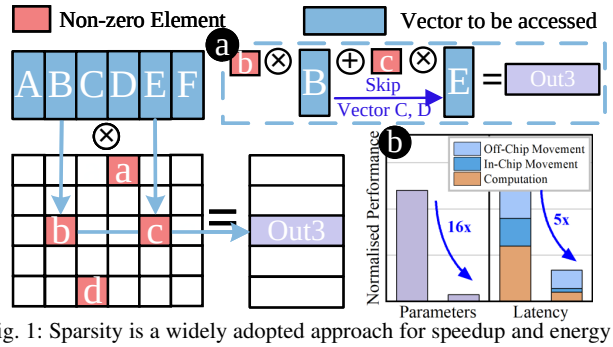


Fig. 1: Sparsity is a widely adopted approach for speedup and energy efficiency through skipping zero in processing, which introduces substantial irregular memory accesses.

To this end, we implement NVR, to the best of our knowledge, the first system-level prefetching mechanism specifically designed for NPUs. Our architecture reduces L2 cache misses by 90% compared to SOTA prefetching techniques in general-purpose processors through the effective prediction of irregular access patterns, with under 5% area overhead. Furthermore, NVR reduces off-chip memory accesses by 75% during NPU execution. When combined with NSB (a small cache structure in NPUs), there is an additional 80% off-chip memory access reduction benefit. Our system-level evaluation of LLM workloads further validates the effectiveness of NVR, demonstrating an average 50% throughput improvement in IO-bound scenarios.

It's worth noting that NVR is not limited to sparse tasks. It can be applied to any operations exhibiting unstructured characteristics, resulting in irregular memory accesses.

II. Background and Motivation

Although sparse workloads offer high theoretical memory-level parallelism (MLP), hardware systems struggle to exploit this algorithmic advantage effectively. As shown in Fig. 1 ⓑ, we systematically profile the parameters of Double Sparsity [4] in LLMs workload and its performance on NPU with a 256KB L2 cache. We find that reducing parameters does not lead to proportional decreases in off-chip memory accesses, resulting in out-of-bounds accesses for explicit buffers (like scratchpad) or cache misses for L2 cache in the chip. As these two scenarios are essentially identical, we focus our analysis on cache misses in the following discussion.

A. Sparse Memory Accesses: Misses Are a Fact of Life

In this section, we introduce the sparse computation and analyse the sparse irregular memory access patterns, which lead to a heavy cache miss and NPU stalls. As an example, Fig. 2 (right) illustrates a typical sparse matrix multiplication (SpMM), which is common in DNNs, computation with compressed sparse row (CSR) format. The code snippet demonstrates the sparse weight matrix (`W`) selectively indexes

* Equal contribution.
✉ Corresponding author.

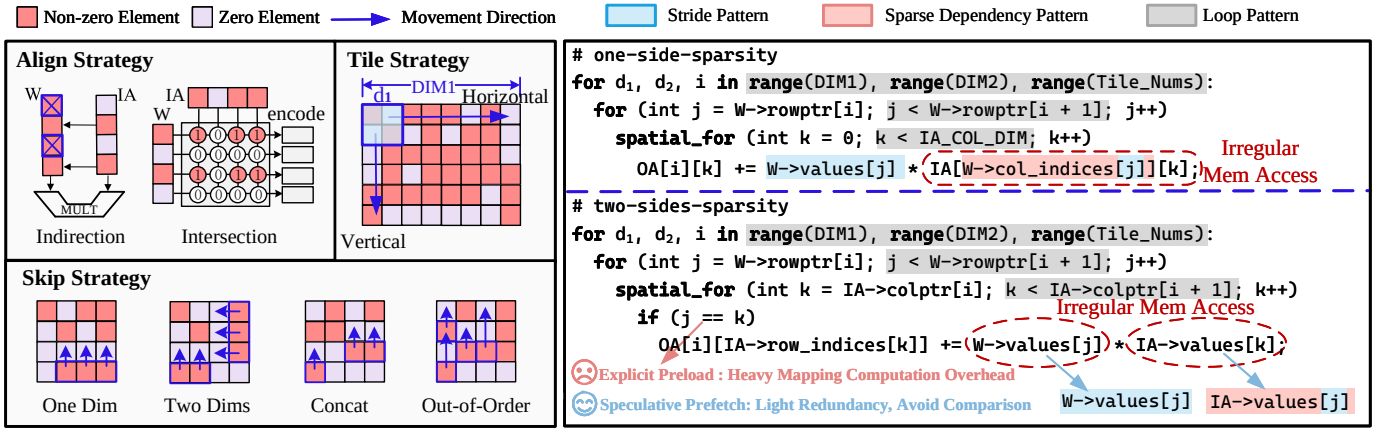


Fig. 2: Sparse Matrix Multiplication can be categorised into one-side-sparsity and two-sides-sparsity patterns, with higher sparsity offering greater speedup potential at the cost of more challenging access patterns. Here, **`spatial_for`** denotes parallel operation on the NPU, while **`IA`** (input activation), **`W`** (weight), and **`OA`** (output activation) represent the input variables, weight parameters, and output results, respectively.

input activation (`IA`) vectors for multiply-accumulate operations. This workload exhibits several critical factors contributing to cache misses:

Indirect Memory Accesses Since only non-zero data is computed, the data must be aligned, which involves numerous index dependency chains. While `W[j]` accesses are often sequential and can be captured by streaming prefetch, `IA[sparse_func(W[i])]` accesses target non-sequential locations. Given that, `IA` typically spans a large index space that exceeds L2 cache capacity, resulting in frequent cache misses. This alignment pattern is common in across sparsity applications, like hash-table indexing and sampling operation in point cloud networks, where `sparse_func` is replaced by complex indexing schemes, often requiring dedicated processing units.

Dynamic Loop Boundaries The tiling strategy refers to reordering element arrangements for hardware computation units (e.g., systolic arrays, vector units). This manifests as irregular loop boundaries. In large dynamic routing architectures such as Mixture-of-Experts (MoE), the memory span between `rowptr[i]` and `rowptr[i+1]` can be substantial, posing significant challenges for cache capacity.

Data Shuffle As shown in Fig. 2 (left), NPUs employ various skip strategies to accelerate computation by bypassing zero elements. NPUs with stronger reordering capabilities can group farther distant data elements for unified processing. This leads to densely packed, long-stride access patterns within short time intervals, causing catastrophic cache behaviour. This issue is particularly evident in LLMs' KVCache [4], where sparse attention employs TopK selection to retain only the k highest-scoring vectors for computation. For LLMs with extended contexts, KVCache indexing can span several gigabytes, significantly exceeding cache capacity.

B. Cache-miss Vulnerable NPUs

Facing numerous cache misses, NPUs suffer even more severe performance degradation than CPUs and GPUs. While CPUs leverage Out-of-Order (OoO) execution and reorder buffers (ROB) to tolerate this through fine-grained instruction-level overlap, and GPUs utilise thread-level scheduling for cache misses hiding, NPUs remain limited in such capabilities. Built primarily on Single Instruction Multiple Data (SIMD) architectures, NPUs possess only coarse-grained instruction parallelism, with compute and memory movement channels decoupled. As demonstrated in Fig. 5 in the experiment, even with ideal OoO execution on NPUs, it also shows suboptimal performance in handling cache misses. Critically, the data-parallel nature of NPUs means a cache miss in any vector element stalls the entire processing pipeline, leading to performance degradation on sparse workloads.

C. Challenge for NPU Prefetch

There are many conventional prefetchers designed for handling irregular memory access patterns mentioned in section II-A. For sparse workloads with abundant irregular memory accesses, simple pattern-based [11]–[13] or history-based [14]–[16] prefetchers often fail to effectively capture and predict these accesses, whereas runahead-based [9], [17], [18] prefetching has been widely adopted as the solution. Runahead employs speculative execution to hide cache miss latency by prefetching future memory accesses. VR [19] and DVR [10] explore leveraging vector units to perform runahead in parallel, which is a well-suited approach for NPUs that inherently support vector operation instructions. However, applying these techniques directly to NPUs presents several challenges.

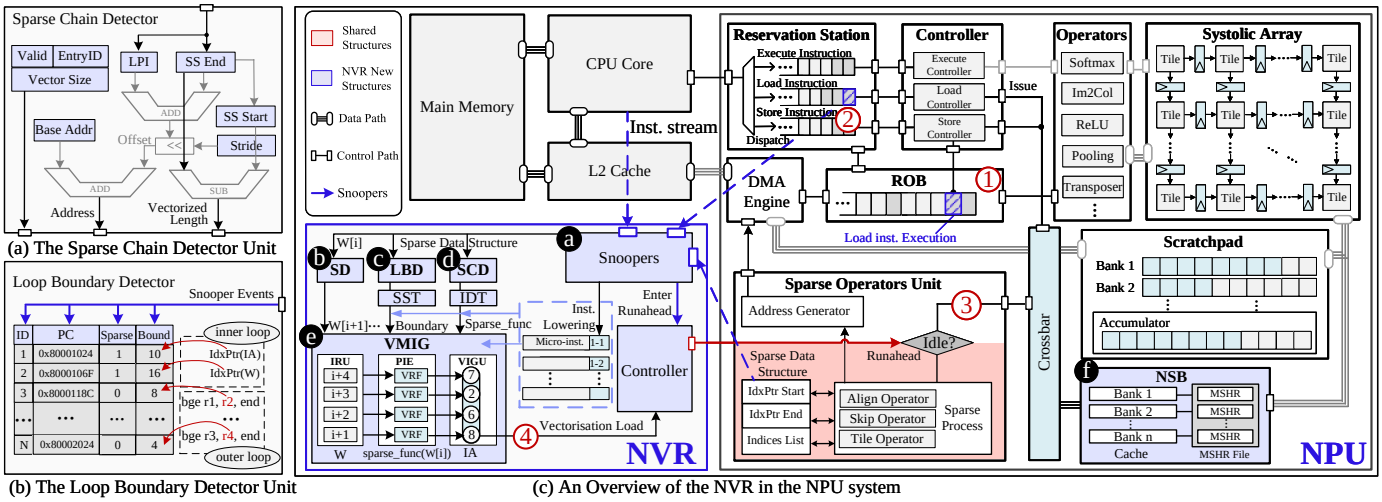
Despite the success of these prefetching techniques in general-purpose processors, the unique characteristics of NPU architectures and their workloads necessitate a fundamentally different approach. Diverse sparse workloads on NPUs often require customised hardware implementations, such as hash-table-based methods in point cloud processing. Meanwhile, sparse workloads commonly leverage specialised sparse data formats, like TACO's [20] multi-dimensional encoding and SMASH's [21] hierarchical bitmap-based decompression, handled by dedicated processing units. This diversity of dependency chains, coupled with the need for parallel execution, poses significant challenges for prefetchers in both pattern capture and overhead. A generalised prefetching approach decoupled from specific sparse computation patterns is necessary. Moreover, NPUs operate with coarse-grained instructions that process entire vectors or matrices. The decoding of such coarse-grained instructions implies weaker locality awareness and significant computational overhead, making traditional instruction-level optimisations less applicable.

III. NVR: Design Philosophy

With these challenges in mind, we propose an adaptation of runahead tailored for NPU architectures, addressing three key aspects:

Decoupled and Non-Invasive Philosophy: Throughput is the primary metric for NPUs. To maximise it, prefetching mechanisms should avoid introducing additional control logic that stalls execution, ensuring maximum data parallelism [22]. Our design decouples from NPU computation logic, allowing simultaneous speculative execution to proactively generate memory requests. The system maintains non-invasive integration by passive monitoring the state and extracting load instruction information through read-only operations. This approach allows early initiation of memory requests without interfering with NPU execution or custom instruction requirements.

Coverage-Oriented Philosophy: In NPU vector operations, computation can proceed only when all data in the batch are ready. Our experiments shown in Fig. 8 (a) reveal that the overall cache miss rate decreases significantly faster than per-batch cache miss rates. While optimising per-batch cache misses is more challenging, it's crucial for substantial performance improvement. Consequently, our



(a) The Sparse Chain Detector Unit

(b) The Loop Boundary Detector Unit

(c) An Overview of the NVR in the NPU system

Fig. 3: NVR micro-architecture and components. Purple blocks represent NVR additions to the system. Red blocks indicate shared components between NVR and NPU, assisting speculative execution during NPU sparse unit idle periods. (b) SD: Stride Detector; (c) LBD: Loop Bound Detector; (d) SCD: Sparse Chain Detector; (e) VMIG: Vectorisation Micro-Instruction Generator; (f) NSB: Non-blocking Speculative Buffer.

design prioritises complete batch retrieval, accepting some prefetch redundancy as a reasonable trade-off. On the other hand, combining this with fuzzy range loading has the added benefit of reducing the overhead of control logic, like branch prediction.

Micro-Instruction-Level Vectorisation Philosophy: NPUs' SIMD architecture operates with vectorised instructions, which typically could be decomposed into multiple micro-instructions spanning several cycles. This fine-grained approach enables precise handling of cache misses while detecting stronger memory access patterns in this granularity. Leveraging NPUs' native support for vector load instructions, we can bundle memory addresses without additional execution units, improving both prediction accuracy and MLP.

In summary, vector runahead on NPU has several unique features. We can summarise our runahead design into three Q&A and introduce detailed micro-architecture in the next session:

Q&A1. When to enter runahead mode: Enter runahead when a load instruction in the NPU's ROB executes, prefetching for the next load instruction in the reservation station. NVR executes in parallel with the NPU instruction stream, extracting information through snooping to generate vectorised memory requests ahead of NPU execution. By computing approximate boundaries in advance, it enables early issuance of vector load instructions.

Q&A2. Where to execute NVR: The NVR is integrated between the CPU and NPU, prefetching speculatively. To achieve decoupled and non-invasive operation, the prefetching logic is architecturally separated from the NPU pipeline.

Q&A3. How to prefetch: NVR leverages idle computational resources in NPU's sparse processing units to perform approximate dependency chain calculations in parallel. Subsequently, NVR generates native NPU vectorised load instructions for the NPU pipeline, prefetching data into the L1/L2 cache hierarchy. This approach efficiently exploits NPU's vector instructions and architectural characteristics for effective prefetching and analysis.

IV. NVR Micro-Architecture

We use Gemmini [23] as our baseline to demonstrate NVR's applicability to general NPUs, as it embodies typical NPU architecture. Its coarse-grained design inherently struggles with cache miss handling, making it ideal for demonstrating NVR's design and effectiveness.

A. Overall Architecture

Fig. 3 illustrates NVR's micro-architectural modifications to the baseline Gemmini configuration, which comprises an in-order core and DNN accelerator sharing a unified L2 cache. The original Gemmini does not support sparse computation, so we have incorporated a basic implementation of the sparse function to demonstrate our prefetcher's design. As illustrated in Fig. 3 (b), the sparse unit is primarily designed to handle alignment, skipping, and tiling of sparse data, three processing techniques discussed in Section II-A.

To support NVR, we augment the baseline with the following structures (purple blocks): A snooper unit monitoring the status of the CPU and the NPU (Fig. 3 (a)). A stride detector (SD) that tracks stream memory access patterns (Fig. 3 (b)). A loop boundary detector (LBD) that performs predictive analysis of iteration bounds for both unrolled and nested loop structures (Fig. 3 (c)). A sparse chain detector (SCD) identifies indirect memory access dependency chains and computes their corresponding memory addresses (Fig. 3 (d)). A vectorisation micro-instruction generator (VMIG) that bundles prefetch sequences with related access patterns into vectorised prefetch operations, optimising bandwidth utilisation (Fig. 3 (e)). An optional non-blocking speculative buffer (NSB), which serves as a small in-NPU cache equipped with miss status holding registers (MSHRs) (Fig. 3 (f)).

The steps of the NVR startup and execution process are numbered with red circles in Fig. 3. The NVR initiates runahead execution upon detecting new load instructions in execution within the NPU's ROB. The NVR's controller monitors CPU and NPU states via snoopers and triggers runahead request for speculative computations during NPU sparse unit idle periods. During execution, the stride detector captures patterns at index i to predict subsequent `W[i]` values. The SCD supports this by generating indirect access address generation for `IA[sparse_func(W[i])]` predictions. To optimise prefetch operations, the LBD analyses loop structures to prevent overruns and enables request packing. The VMIG reconstructs decomposed micro-instructions by synthesising information from these components and generates new vectorised prefetching instructions inserted into the NPU pipeline for prefetching operations. As a complementary approach, the NSB offers an optional mechanism to enhance the prefetching strategy, demonstrating a reduction in cache misses experimentally. The following sections detail these components' design.

B. Stride Detector (SD)

The stride detector is the fundamental pattern recognition unit in NVR design and primarily used to predict the next batch of addresses for `W[i]`. It continuously predicts the data memory access pattern for step patterns by stream prefetching. For `W[i]`, neural networks tend to have relatively few branch predictions in their workloads, and

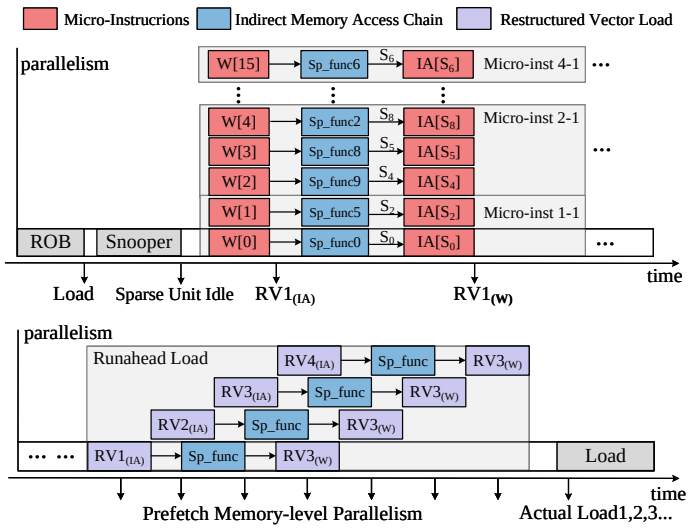


Fig. 4: Vectorisation micro-instruction generation pipeline. Micro-instruction 1-1 represents the first micro-instruction of instruction 1. Each micro-instruction loads an indeterminate number of data.

a high prediction accuracy can be achieved by recording the address differences and the confidence levels.

C. Snoopers and Controller

The snoopers are non-invasive probes used to precisely extract the architectural states of both the CPU and NPU. We monitor three critical signal types: (1) branch instructions from the CPU, which provide LBD with nested loop context information, (2) custom load-related instructions in the NPU, which are used to determine the optimal timing for runahead mode activation, and (3) NPU sparse unit registers, which supply the metadata essential for the NVR prediction mechanisms. The read-only, non-invasive design of the snoopers maintains architectural integrity by preventing any modifications to NPU computational logic and avoiding interference with the interactions between the CPU and NPU. Upon entering runahead mode, the controller sends speculative execution requests to the sparse unit. When runahead mode is triggered, the controller sends speculative execution requests to the sparse unit, enabling it to proceed with predictive computations. The system then monitors the unit availability, using the snooper infrastructure to retrieve the necessary sparse unit data as soon as it becomes available.

D. Sparse Chain Detector (SCD)

The SCD identifies and predicts patterns in sparse computations through two critical components: historical information preservation and indirect prefetching pattern learning. An Indirect Pattern Table (IPT) is maintained to record indirect prefetch patterns, including details such as the Last Prefetch Indirect (LPI) and the sparse structure's start address. In DNN workloads, computation index patterns typically exhibit locality characteristics. Due to the large volume of data processing, indirect index patterns remain relatively stable over time intervals, often appearing as shallow indirect chains, as formulated below:

$$\mathrm{IA}_{\mathtt{address}} = \mathrm{IA}_{\mathtt{ss_start}} + (\mathrm{W}_{\mathtt{LPI}} << \mathtt{stride})$$

, where $\mathrm{IA}_{\mathtt{address}}$ represents the predicted indirect access address, $\mathrm{IA}_{\mathtt{ss_start}}$ denotes the base address at the start element of the loop iteration, and $\mathrm{W}_{\mathtt{LPI}}$ indicates the value of last prefetched W. Here, `stride` defines the offset between consecutive memory accesses. The prediction for the address of IA depends on the value of $\mathrm{W}_{\mathtt{LPI}}$.

By recording the structural information of `W` and `IA` from previous operations, SCD can effectively track sparse operation chains and predict subsequent indirect prefetching addresses via stride learning. This predictive mechanism is particularly beneficial in scenarios with repetitive sparse patterns. The mechanism requires sparse processing boundaries and current processing indices, such as CSR `col_indices` which locates the non-zero elements in columns and `rowptr` for tracking row start positions from NPU operations, information that is readily available in most sparse data formats. Unlike traditional prefetchers, which lack access to NPU register-level information and often depend on statistical methods like EWMA [24], leading to inefficiencies, our approach uses the NPU's sparse unit to run ahead. This allows us to preserve and leverage historical information, significantly improving the prediction of sparse vector accesses while reducing redundant memory requests.

E. Loop Bound Detector (LBD)

DNN workloads inherently consist of multiple nested and unrolled loops, primarily involving matrix and vector operations, where higher-dimensional operations naturally require deeper loop nesting structures. To address the challenges of memory access patterns in these structures, the LBD employs systematic loop behaviour and boundary tracking to optimise prefetch sizing while preventing boundary-crossing invalid prefetches.

As illustrated in the listing of Fig. 2, nested loops typically occur during the traversal of rows and columns of a matrix, while unrolled loops are often used in parts of matrix multiplication where multiple indirect chains are executed in parallel. As depicted in Fig. 3 (b), LBD maintains historical information through a Sparse Structure Table (SST), where each parallel port has a dedicated entry to facilitate concurrent tracking of sparse chains. LBD implements a dual-mode boundary prediction, handling both sparse and normal boundaries. The mechanism distinguishes loop hierarchies from inner to outer levels through entry IDs, with each entry maintaining comprehensive loop instruction information, including PC value, boundary values, and operational mode. For standard loop boundaries (illustrated in line 1, Fig. 2 (right)), LBD captures historical boundary information by monitoring register values of jump instructions, as shown with RISC-V B-type branch instructions in Fig. 3 (b). For variable boundaries in sparse computations (shown in line 2, Fig. 2 (right)), boundary information is dynamically acquired through snoopers from sparse unit registers. Within this framework, upon detecting loop instructions, the LBD not only uses historical data to predict the boundary but also learns loop boundaries by examining input register values from comparison instructions. At the same loop level, instructions are consolidated into memory access requests, while boundary values act as crucial constraints to prevent excessive prefetching.

F. Vectorisation Micro-Instruction Generator (VMIG)

In NPU-based sparse data processing, the efficiency of the SIMD pipeline is often compromised because traditional vector instructions fail to fully utilise memory bandwidth when dealing with skipped or zero-valued elements. Strictly following NPU runtime loading instructions for prefetching would lead to significant bandwidth wastage. Instead, prefetch operations can bypass the rigid loading logic of the NPU, and the VMIG addresses this by restructuring the NPU's native load instructions at the micro-instruction level. This approach optimises sparse data patterns, leveraging existing vector load units without additional hardware support.

VMIG implements a three-stage pipeline where each stage is executed by dedicated hardware units. Initially, the Instruction Reconstruction Unit (IRU) manages the micro-instruction restructuring, using the SST provided by the LBD and integrating stride detection for speculative execution. The following stage is handled by the Parallel Inference Engine (PIE), which uses the IPT from the SCD and the Vector Register File (VRF) to predict `sp_func` concurrently across multiple data streams. As shown in Fig. 4, it parallelly executes sixteen consecutive dependency chains that generate W from IA. Finally, the Vector Instruction Generation Unit (VIGU) synthesises these restructured load instructions into a single vector operation, dynamically optimising the memory access.

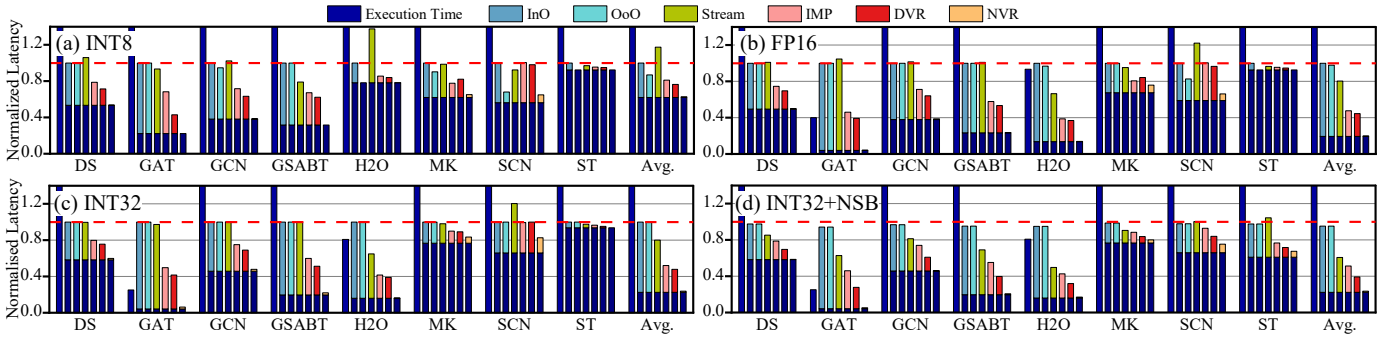


Fig. 5: Normalised wall-clock time latency for each sparse workload. Within each group, each bar from left to right denotes execution in density, in order execution, OoO execution, IMP, DVR, and NVR, respectively. The lower segment indicates the NPU base execution time, whilst the upper segment represents the stall time caused by cache misses.

This optimisation strategy leverages the NPU's inherent SIMD load capabilities and vector registers, eliminating the need for additional vector hardware. As shown in Fig. 4, the process is fully pipelined, enhancing MLP through rapid consecutive memory request issuance. Compared to executing single instructions, this pipelined approach significantly reduces memory access latency and improves bandwidth utilisation. The efficiency also depends on the MSHR, which prevents cache miss events from blocking subsequent prefetch operations.

G. Non-blocking Speculative Buffer (NSB)

For discrete data structures, utilising scratchpad memory incurs substantial logic overhead for data transformation and retrieval operations. However, the characteristics of sparse data patterns present unique opportunities for cache-based optimisation. By strategically storing sparse discrete data in the cache while maintaining continuous data in scratchpad memory (e.g., dense vectors in one-side-sparsity), we can leverage implicit cache line reuse patterns. This cache-based approach naturally accommodates irregular reuse patterns without the costly pre-computation overhead required by scratchpad implementations. To exploit these characteristics, we introduce NSB, a compact non-blocking cache architecture optimised for discrete element management. Sparse workloads exhibit irregular memory access patterns with extensive index spaces at high sparsity, direct-mapped or low-associativity configurations frequently encounter conflict misses. Thus, we implement a high-way set-associative mapping strategy. While NSB cannot mitigate L2 cache misses during prefetch operations (as data inherently resides off-chip), it significantly reduces NPU-to-L2 latency and off-chip memory accesses during actual load instruction execution. Experimental results demonstrate that NSB further reduces bandwidth requirements by 5× compared to baseline NVR prefetching. The design incorporates an MSHR file to manage concurrent memory operations, enabling tracking of outstanding load buffer requests and cache misses. This MSHR infrastructure coalesces multiple outstanding requests to the same cache line, eliminating redundant memory accesses and optimising bandwidth utilisation.

H. Hardware Overhead

We implement NVR in RTL and synthesise our design using Synopsys Design Compiler on TSMC 28nm process technology at 2.0 GHz. The area overhead is 3% and 4.6% relative to the baseline Gemmini architecture, for configurations without and with NSB, respectively. These results demonstrate that NVR achieves its performance benefits with minimal hardware cost.

V. Evaluation

A. Experimental Setup

We implement and evaluate NVR by integrating it into a Gemmini-like NPU model constructed using the ScaleSim simulator [25]. Additionally, we leverage LLMCompass [26], a specialised simulator designed to evaluate hardware optimisations in LLM inference.

TABLE I: Sparse Computation Workload.

Workload	Short	Domain
Double Sparsity [4]	DS	large language model
Graph Attention Networks [35]	GAT	graph neural networks
Graph Convolutional Networks [36]	GCN	graph neural networks
Graph Sparse Attention [37]	GSABT	sparse attention
Heavy-Hitter Oracle [38]	H2O	large language model
MinkowskiNet [39]	MK	point cloud
SparseConvNet [40]	SCN	point cloud
Switch Transformer [41]	ST	mixture of experts

Comparison We evaluate our approach against in-order Gemmini (serial execution of load and compute instructions), ideal out-of-order Gemmini (overlapping the load and computation time), and several prefetchers. We evaluate the following prefetch techniques: 1) stream prefetcher [27], the simplest prefetch mechanism based on stride; 2) IMP [28], one of the SOTA indirect memory prefetchers, prefetching indirect memory access at the L1 cache; 3) DVR [19], one of the SOTA runahead techniques, vectorising the same chain of indirect memory accesses across multiple invocations of an inner loop. 4) Our NPU Vector Runahead. For a fair comparison, we add the Gemmini custom ISA support for these prefetchers and expand them to the same number of parallels as the NVR.

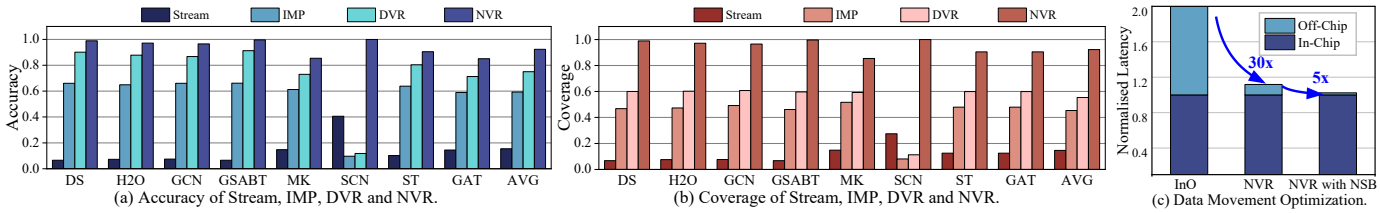
Workload Sparsity is widely present across various workloads in DNNs. We select several key sources of sparsity, including attention mechanisms [29], [30], MoE structures [31], [32], 3D point cloud processing [33], [34], and so on. Table I presents representative workloads extracted from various models' linear layer memory access patterns based on their functionalities and domains.

B. Performance Breakdown

Since DNNs commonly employ low bit-width computations, we evaluate different bit-width configurations (INT8, FP16, INT32) to analyse their impact on cache behaviour and system performance, as shown in Figs. 5 (a), (b), and (c). Higher data bit-width inherently requires longer data movement time and larger cache block capacities, increasing the probability of cache misses.

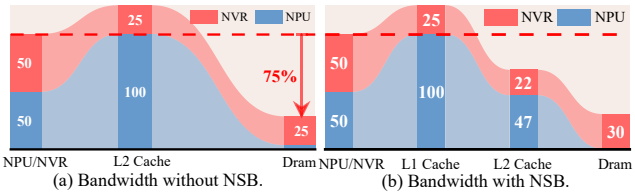
Overall, cache miss stalls represent a substantial portion of wall-clock time across most sparse workloads. The Switch Transformer presents a notable exception, showing lower cache miss ratios due to its relatively fixed network architecture and block-like data distribution patterns. For sparse workloads, the NPU's OoO (out-of-order) execution capabilities struggle to effectively overlap data movement with computation to mitigate cache miss latency, as several workloads are inherently IO-bound. While prefetching mechanisms generally reduce cache miss stall time effectively, stream prefetchers occasionally introduce performance penalties due to their lower accuracy. Notably, our NVR implementation achieve reductions in latency, averaging 98.3%, 99.2%, and 97.3% for INT8, FP16, and INT32, respectively.

As shown in Fig. 5 (a), we evaluate INT32 latency variations with NSB enabled. we evaluate latency variations with NSB using INT32 configurations. The red line represents the baseline latency without NSB. With NSB enabled, L2 cache miss latency decreased by about



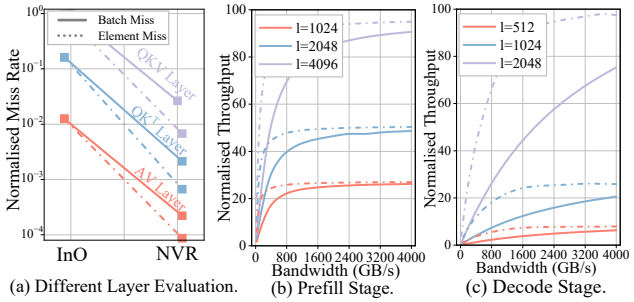
(a) Accuracy of Stream, IMP, DVR and NVR. (b) Coverage of Stream, IMP, DVR and NVR. (c) Data Movement Optimization.

Fig. 6: Prefetcher accuracy and coverage evaluation and the data movement optimisation results of NVR.



(a) Bandwidth without NSB. (b) Bandwidth with NSB.

Fig. 7: Normalised bandwidth allocation without/with NSB.



(a) Different Layer Evaluation. (b) Prefill Stage. (c) Decode Stage.

Fig. 8: System-level Evaluation on LLMs. In (b) and (c), dashed and solid lines represent NVR and baseline performance respectively, where l denotes input/output sequence length.

15%. When combined with NVR, NSB achieves a 40% reduction in cache miss stalls, but performs poorly with conventional stream prefetchers. Thus, NSB activation depends on prefetcher accuracy.

C. Prefetching Effectiveness and Off-Chip Bandwidth Reduction

This experiment evaluates prefetcher performance across sparse workloads using two key metrics: accuracy and coverage. NVR demonstrates consistently high performance, maintaining both accuracy and coverage rates above 90% across most workloads. As shown in Fig. 6 (b), coverage, which is more crucial for NPU performance, presents a greater challenge than accuracy. Benefiting from our fuzzy prefetch strategy, NVR also achieves significantly higher performance in this metric compared to other prefetchers.

The combined effect of accuracy and coverage directly influences prefetch bandwidth and the volume of off-chip memory accesses caused by cache misses during NPU execution. Figs. 7 (a) and (b) demonstrate the total bandwidth, including prefetch bandwidth. In both cases, compared to the InO baseline, the off-chip memory bandwidth is reduced by around 75%. Since the prefetch bandwidth is often more easily overlapped, Fig. 6 (c) shows the NPU actual load execution time with the memory accesses bandwidth removed. NVR can effectively reduce the off-chip memory accesses by 30x, and with NSB, it can be further reduced by 5x.

D. LLM Inference Evaluation

To validate our system-level optimisations, we evaluate the impact of NVR on the performance of LLMs. The experiments are conducted using the LLMCompass simulator. As shown in Fig. 8 (a), the NVR implementation reduces overall latency by decreasing cache miss stall time in the LLMs. Under NVR, both overall and per-batch cache miss rates decrease exponentially, with the latter showing a slower decay rate. LLMs are typically composed of two main stages: prefill and decode. It has been proven that the prefill stage is compute-bound, while the decode stage is IO-bound. Figs. 8 (b) and (c) demonstrate the benefits of applying NVR to the entire LLM network.

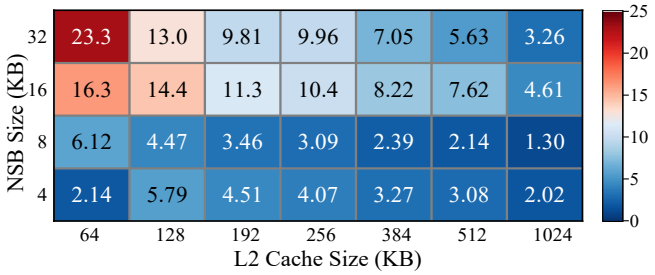


Fig. 9: NSB and L2 cache impact on performance. Performance metric: the inverse of latency and area (higher is better).

It is evident that for the compute-bound prefill stage, NVR can expedite reaching maximum throughput, particularly in low-bandwidth scenarios. For the IO-bound decode stage, the NVR architecture demonstrates an average 50% throughput improvement through reduced off-chip memory accesses. This enhancement becomes increasingly pronounced with longer output sequences.

E. Sensitivity Analyses

To evaluate NSB's efficacy, we conduct comprehensive sensitivity analyses across varying NSB and L2 cache parameters. The performance metrics represent the inverse relationship between latency and area, calculated as the product of NSB and L2 Cache dimensions, ensuring equivalent area penalties for both scaling approaches. As illustrated in Fig. 9, modest NSB buffer configurations yield substantially higher performance improvements than equivalent L2 cache scaling. Specifically, in a configuration with 256KB L2 cache and 4KB NSB, quadrupling the NSB capacity delivers 5x the performance benefit over scaling the L2 cache to 1024KB.

VI. Conclusion

In this paper, we present NVR, a runahead prefetching mechanism addressing the challenges of sparse DNN workloads on NPUs. This work demonstrates specialised prefetching techniques for DNN applications on custom architectures beyond general-purpose processors. **Lessons we learnt.** This work adopted a holistic, workload-driven approach to accelerate the DNNs' executions. By workload profiling and architectural analysis, key bottlenecks were identified, guiding targeted system re-architecture. Instead of simply adding more hardware resources, this work addressed fundamental inefficiencies, achieving performance gains equivalent to multiple times the benefits of hardware scaling, but with significantly reduced area and power costs. The constructed NVR provides key insights that precise, interdisciplinary optimisations rooted in a deep understanding of workload-architecture interactions. We believe that such analysis offers valuable insights into bridging architectural and machine learning research, fostering more effective and sustainable design strategies.

VII. Acknowledgment

We appreciate the reviewers for their helpful feedback. This work is supported by the National Key Research and Development Program (Grant No. 2024YFB4405600), the National Natural Science Foundation of China (Grant No. 62472086), the Basic Research Program of Jiangsu (Grant No. BK20243042), the Science and Technology Major Special Program of Jiangsu (No. BG2024010), and the Start-up Research Fund of Southeast University (Grant No. RF1028624005).

REFERENCES

[1] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.

[2] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[3] Z. Yao, S. Cao, W. Xiao, C. Zhang, and L. Nie, "Balanced sparsity for efficient dnn inference on gpu," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 5676–5683.

[4] S. Yang, Y. Sheng, J. E. Gonzalez, I. Stoica, and L. Zheng, "Post-training sparse attention with double sparsity," *arXiv preprint arXiv:2408.07092*, 2024.

[5] F. Farshchi, Q. Huang, and H. Yun, "Integrating nvidia deep learning accelerator (nvdla) with risc-v soc on firesim," in *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE, 2019, pp. 21–25.

[6] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[7] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.

[8] S. Liu, P. Li, J. Zhang, Y. Wang, H. Zhu, W. Jiang, S. Tang, C. Chen, Q. Liu, and M. Liu, "16.2 a 28nm 53.8 tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 250–252.

[9] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead execution: An alternative to very large instruction windows for out-of-order processors," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.* IEEE, 2003, pp. 129–140.

[10] A. Naithani, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Vector runahead," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 195–208.

[11] N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture*, 1990, pp. 364–373.

[12] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 63–74.

[13] P. Michaud, "Best-offset hardware prefetching," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 469–480.

[14] Y. Ishii, M. Inaba, and K. Hiraki, "Access map pattern matching for data cache prefetch," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 499–500. [Online]. Available: https://doi.org/10.1145/1542275.1542349

[15] S. Somogyi, T. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *33rd International Symposium on Computer Architecture (ISCA'06)*, 2006, pp. 252–263.

[16] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 141–152.

[17] M. Hashemi, O. Mutlu, and Y. N. Patt, "Continuous runahead: Transparent hardware acceleration for memory intensive workloads," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[18] A. Naithani, J. Feliu, A. Adileh, and L. Eeckhout, "Precise runahead execution," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 397–410.

[19] A. Naithani, J. Roelandts, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Decoupled vector runahead," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 17–31.

[20] P. A. Tew, "An investigation of sparse tensor formats for tensor libraries," M.Eng. Thesis, Massachusetts Institute of Technology, Cambridge, MA, Jun 2016. [Online]. Available: http://tensor-compiler.org/files/tew-meng-thesis-sparse.pdf

[21] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppula, N. M. Ghiasi, T. Shahroodi, J. G. Luna, and O. Mutlu, "Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 600–614.

[22] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 37–47.

[23] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 769–774.

[24] S. V. Crowder and M. D. Hamilton, "An ewma for monitoring a process standard deviation," *Journal of Quality Technology*, vol. 24, no. 1, pp. 12–21, 1992.

[25] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[26] H. Zhang, A. Ning, R. B. Prabhakar, and D. Wentzlaff, "Llmcompass: Enabling efficient hardware design for large language model inference," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1080–1096.

[27] I. Hur and C. Lin, "Memory prefetching using adaptive stream detection," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 2006, pp. 397–408.

[28] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "Imp: Indirect memory prefetcher," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 178–190.

[29] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021.

[30] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 53–68, 2021.

[31] Z. Chen, Y. Deng, Y. Wu, Q. Gu, and Y. Li, "Towards understanding mixture of experts in deep learning," *arXiv preprint arXiv:2208.02813*, 2022.

[32] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat *et al.*, "Glam: Efficient scaling of language models with mixture-of-experts," in *International Conference on Machine Learning*. PMLR, 2022, pp. 5547–5569.

[33] M. Gadelha, R. Wang, and S. Maji, "Multiresolution tree networks for 3d point cloud processing," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 103–118.

[34] A. V. Phan, M. Le Nguyen, Y. L. H. Nguyen, and L. T. Bui, "Dgcnn: A convolutional neural network over large-scale labeled graphs," *Neural Networks*, vol. 108, pp. 533–543, 2018.

[35] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[36] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[37] D. Zhang, J. Yan, K. Polat, A. Alhudhaif, and J. Li, "Multimodal joint prediction of traffic spatial-temporal data with graph sparse attention mechanism and bidirectional temporal convolutional network," *Advanced Engineering Informatics*, vol. 62, p. 102533, 2024.

[38] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 34 661–34 710, 2023.

[39] S. Brahmbhatt, V. Mehta, and S. Ghosh, "Minkowski sar-unet3d for point cloud semantic segmentation."

[40] J. Wang, W. Li, M. Zhang, and J. Chanussot, "Large kernel sparse convnet weighted by multi-frequency attention for remote sensing scene understanding," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 61, pp. 1–12, 2023.

[41] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

Towards Training Robustness Against Dynamic Errors in Quantum Machine Learning

Shijin Duan[1], Gaowen Liu[2], Charles Fleming[2], Ramana Kompella[2], Xiaolin Xu[1], Shaolei Ren[3]
[1]*Northeastern University*, [2]*Cisco Research*, [3]*University of California, Riverside*
{duan.s, x.xu}@northeastern.edu, {gaoliu, chflemin, rkompell}@cisco.com, shaolei@ucr.edu

***Abstract*—Quantum machine learning, crucial in the noisy intermediate-scale quantum (NISQ) era, confronts challenges in error mitigation. Current noise-aware training (NAT) methods often assume static error rates in quantum neural networks (QNNs), overlooking the dynamic nature of quantum noise. Our work highlights how error rates fluctuate over time and across different qubits, affecting QNN performance even when overall error rates are similar. We introduce a novel NAT strategy that dynamically adjusts to standard and fatal error conditions, incorporating a low-complexity search method to identify fatal errors during optimization. This strategy significantly improves robustness, maintaining competitive performance with leading NAT methods across varying error scenarios.**

I. Introduction

Quantum machine learning (QML) is emerging in various practical applications [1]–[3]. Specifically, the variational quantum algorithm (VQA) [4] is promising to solve near-term QML tasks. The VQA is performed in the hybrid classical-quantum optimization scenario, where parameterized quantum circuits (PQCs), a.k.a. *ansatz*, produce measurements of quantum states on the quantum computer, and the classical computer performs parameter optimization of PQCs through gradient-based methods [5]. We demonstrate this procedure in Fig. 1, taking qauntum neural network (QNN) as an example.

However, the execution of PQCs faces rigorous situations with noise. As quantum computers move to the noisy intermediate-scale quantum (NISQ) era, errors in qubits and quantum operations are inevitable due to the dynamic nature of the quantum system and its environment [6]. Thus, error mitigation in PQCs is critical for robust execution on quantum computers, where mitigation strategies have been proposed during QNN training [7] or execution [8]–[10]. In this work, we focus on noise-aware training (NAT) for QNN models, aiming to enhance the robustness of PQC against errors. The current NAT approach proposed by [7] involves deliberately introducing simulated noise into quantum gates, based on a predefined error rate. Despite the pronounced effectiveness, we highlight a critical limitation in the current NAT strategy and PQC simulators, which rely on static error rates during noise analysis. ***In reality, the noise landscape is dynamic across both time and qubits.*** Recent study demonstrates that quantum noise levels can fluctuate significantly over time, even within a transient time window [11]. We further emphasize the dynamic nature of noise across different qubits. Such dynamic noise landscape renders the static error rate assumption in NAT insufficient for practice. For instance, a qubit in simulation that is assumed to have a low error rate could be mapped to a physical qubit with a significantly higher error rate.

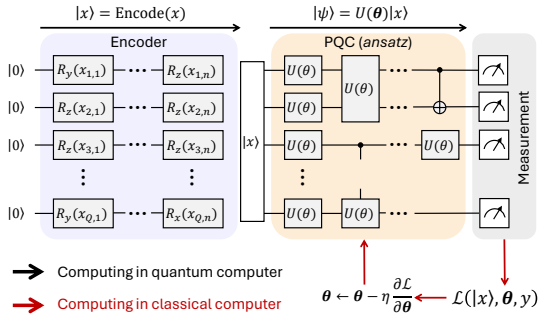


Fig. 1: A demonstration of QNN task. Input x is encoded as a state vector $|x\rangle$ using rotation gates [12], which is fed into PQC to prepare a final state vector $|\psi\rangle$ for measurement.

Additionally, current approaches to evaluating errors in noisy QNN models predominantly focus on the probability of errors, rather than assessing the actual detrimental impact these errors have on QNN performance. In practice, even errors with identical probabilities can have vastly different consequences. Certain errors may have a negligible influence on the QNN's inference performance, while some fatal errors can degrade its performance to levels worse than random guessing. The situation is more critical when qubits with possible fatal errors are mapped to high-error-rate physical qubits, posing a significant risk to the robustness of QNN inference.

In this paper, we characterize the influence of dynamic noise on PQCs, and demonstrate the scenarios where state-of-the-art (SOTA) NAT strategy is not competent. Our analysis underscores the need for a more comprehensive NAT strategy that transcends static error rate consideration and effectively mitigates fatal errors. To tackle this challenge, we propose a low-complexity search strategy. It can search for fatal errors along with QNN training, and evolve with PQC parameters simultaneously. Rather than randomly sampling errors, we utilize the searched fatal errors to boost the robust QNN training. We highlight our contributions as follows:

- We reveal the significant influence of dynamic noise landscape on QNN models. We also quantify the negative influence of error cases on QNN models, and propose the analysis methods for the fatal errors on a QNN model.
- To the best of our knowledge, this is the first NAT method

that is independent of error rates, but concentrates on fatal errors of PQCs. Our strategy aims to develop QNN models adaptive to various noise distributions and with good tolerance to these fatal error cases.
- We evaluate the effectiveness of our strategy on various QML tasks, which greatly improves the robustness of QNN models against fatal errors. Also, our strategy can achieve performance comparable to that of the SOTA NAT strategy under various error rates.

II. Preliminary

A. Quantum Neural Network with Noise

As illustrated in Fig. 1, a QNN model starts with a predefined encoder [7], [12], [13] to prepare the query x as a quantum state vector $|x\rangle$. It then applies a sequence of parameterized unitary gates $U_g(\theta_g) \in \mathcal{G}$ represented by unitary matrix $U(\boldsymbol{\theta}) = \prod_{g=G}^{1} U_g(\theta_g)$. The final quantum state is denoted as $|\psi\rangle = U(\boldsymbol{\theta})|x\rangle$. Then, the measurement of PQC is tackled by evaluating an observable B on $|\psi\rangle$, which is expressed as

$$f(|x\rangle, \boldsymbol{\theta}) = \langle\psi| B |\psi\rangle = \langle x| U^{\dagger}(\boldsymbol{\theta}) B U(\boldsymbol{\theta}) |x\rangle \quad (1)$$

In the following, we use $f(|x\rangle, \boldsymbol{\theta})$ to denote the expected measurement results of PQC.

Noise Model. To simulate noise effects in PQCs, we consider quantum errors such as depolarizing noise and phase damping. They can be generally represented as `Pauli-X`, `Pauli-Y`, and `Pauli-Z` operation after gates, as outlined in [7]. Developing QNNs with inherent resilience to Pauli errors during training can significantly alleviate extensive error correction during execution. Additionally, post-training error mitigation methods, such as measurement error mitigation [9], have been widely explored and orthogonal to our approach. This combination is further explored in Sec. V-A.

Deterministic Modeling. The deterministic noise model estimates the PQC output in certain error cases [14]. Assuming an error E_g (as an operation) could occur on the gate $U_g(\theta_g)$, e.g., $U_g(\theta_g|E_g) = E_g U_g(\theta_g)$, we define the measurement as

$$f(|x\rangle, \boldsymbol{\theta}|\boldsymbol{E}) = \langle\psi| B |\psi\rangle, \text{ where } |\psi\rangle = U(\boldsymbol{\theta}|\boldsymbol{E})|x\rangle \quad (2)$$

The noisy PQC conducts $U(\boldsymbol{\theta}|\boldsymbol{E}) = \prod_{g=G}^{1} E_g U_g(\theta_g)$ on $|x\rangle$, where $\boldsymbol{E} \in \{I, X, Y, Z\}^G$ is a specific error case. Deterministic modeling tackles the analysis of error effect under certain errors, such as systemetic errors [15] and coherent errors [16]. In our work, we utilize this model to evaluate the worst-case errors for PQC measurement.

B. Related Work

The noise-aware training (NAT) on QNNs has been recently proposed to apply robust training strategies during parameter optimization [7]. Specifically, random noise injection is utilized during training, and other strategies, such as normalization and quantization, are applied as post-training robustness enhancement. Our work aligns with the in-training methodology, for example, noise injection, while it is orthogonal to the other strategies. The ensemble strategy is also proposed to improve performance with majority voting [17]; however, the multiplied model size will either burden the quantum resources or lengthen the execution time along the ensemble scale. Nevertheless, both works assume a static and unique noise distribution during QNN training, which contradicts reality, as quantum noise exhibits dynamic properties.

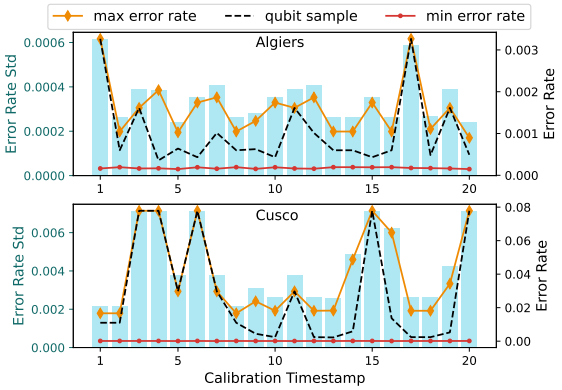


Fig. 2: Qubit error rates on both qubit scale (left y-axis) and time scale (right y-axis). We record the `Pauli-X` gate error rates of all qubits of two IBM quantum computers, `ibmq_algiers` (27 qubits) and `ibmq_cusco` (127 qubits).

C. Scope of Our Work

Application Scenario. We aim to mitigate the dynamic error effect on quantum computers by training a robust model for *future use*. It should perform well under dynamic and unpredictable noise. Therefore, QNN models pre-trained through classical-quantum hybrid algorithms are ideal for this scenario. Other VQA tasks, such as VQE [18], fall outside our scope. For instance, once the state is optimized and prepared in VQE, no further execution of the PQC is needed. In such tasks, real-time error mitigation methods, including surface codes [19] and iterative skipping strategies [11], are more appropriate.

Case Study. In the following analysis, validation is provided as a numerical proof of concept along with the analytical dicussion. Without loss of generality, we take a case study from a 4-qubit QNN, whose PQC contains 3 layers. Each layer is composed of 4 `U3` gates and 4 `CU3` gates in a cyclical manner [12]. This model is trained for an MNIST-2 task that classifies digits 3 and 6.

III. Dynamic Noise and Fatal Error on QNN

A. Severe Error Variation on Time Scale and Qubit Scale

In previous NAT on QNN, the model parameters are usually optimized under the static noise assumption. However, transient errors have recently been recognized for both long-term and short-term quantum operations [11], where the characteristic of qubits could vary significantly along the ***time*** scale.

Even worse, large variations of qubit characteristics also exist along the ***qubit*** scale. In Fig. 2, we demonstrate the dynamic noise landscape from two IBM commercial quantum computers, on the time scale and the qubit scale. During a long-term run, the gate error rate on a certain qubit (e.g., shown as

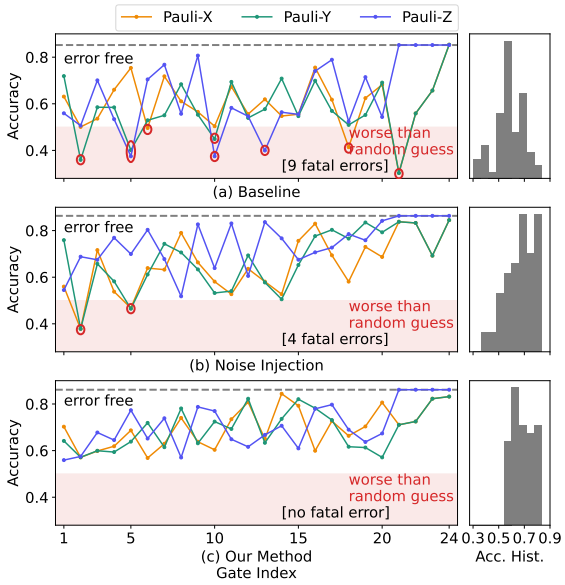


Fig. 3: The inference accuracy of QNN, trained by (a) no error consideration (b) noise-injection strategy and (c) our method (Sec. IV), for the MNIST-2 task, assuming only one gate occurring logical error. Right plots are the corresponding histograms of accuracies.

black dash lines in Fig. 2) could vary by an order of magnitude in the worst case. On the qubit scale, the error rates of different qubits also have a significant variation. In some extreme cases, the error rate on one qubit can be 80 times greater than that on another. Thus, for a well-trained QNN, different mappings from its qubits to the physical qubits on quantum computers can induce significant error rate divergence on each qubit.

B. Fatal Errors Poisoning QNN Accuracy

Besides the error rate variation, not all gates in a PQC have the same sensitivity to errors. We present this assessment of QNN in Fig. 3, under a minimal error scenario where only one single gate is affected by error. The result is intriguing, since even one gate error can significantly compromise the QNN model into a useless level. Specifically, there are 9 out of 72 errors, for a QNN trained without noise consideration, that result in performance below random guess. Even with SOTA noise injection training [7], there are still 4 errors that lead to worse than random guess. This underscores that errors at different locations can cause widely disparate levels of performance degradation, even with similar error rates. On real quantum computers, the occurrence of such a fatal error can render the QNN inference utterly unreliable. Under the dynamic noise, the coming of fatal error cases is also unpredictable.

As a preview, our strategy significantly enhances QNN performance across all error cases, ensuring that predictions consistently exceed random guesses in all scenarios explored in our case study. Furthermore, accuracy histograms indicate that SOTA noise injection training generally improves model accuracy under most error conditions over the error-free training (baseline). However, this approach falls short in addressing specific fatal error scenarios effectively. In contrast, our method successfully mitigates the impact of these fatal errors without compromising model performance in other error scenarios.

C. Worst Case Analysis

Evaluating all possible error scenarios during QNN training is both time-consuming and impractical, especially since many of these scenarios have negligible probabilities. Instead, it is more meaningful to focus on error cases with relatively high probabilities, as these are the most common during quantum execution. For example, with an average error rate of 0.001 (such as in `ibmq_cusco`) across a 100-gate PQC, the conditional probability that one or two errors occur given the error occurring, is $P(N_{\mathcal{E}} \leq 2 | N_{\mathcal{E}} \geq 1) \approx 0.9984$ in normal use. Therefore, we emphasize concentrating on these "high-probability" errors, since they are the most likely to affect the model's performance when errors occur. In our analysis, we locate fatal errors in the high-probability error set and utilize the deterministic noise model described in Eq. 2.

Fatal loss $\mathcal{L}_{sup}$. Given an error set $\Omega_{\boldsymbol{e}}$ and input-label pair (x, y), we define the fatal loss to $f(|x\rangle, \boldsymbol{\theta})$, as the supremum of losses caused by all $\boldsymbol{E} \in \Omega_{\boldsymbol{e}}$,

$$\mathcal{L}_{sup}(\Omega_{\boldsymbol{e}}, x, y) = \sup_{\boldsymbol{E} \in \Omega_{\boldsymbol{e}}} \{\mathcal{L}(f(|x\rangle, \boldsymbol{\theta} | \boldsymbol{E}), y)\} \tag{3}$$

Accordingly, the QNN fatal accuracy evaluates the worst-case performance of PQC measurement against all noise cases in $\Omega_{\boldsymbol{e}}$. Our metric in Eq. 3 does not involve gate error rates during evaluation; thus, the evaluation remains fair and robust against dynamic noise changes in quantum computers.

Error Set Collection. The "high-probability" $\Omega_{\boldsymbol{e}}$ is collected from typical error cases observed under normal operating conditions, given the scale of the PQC and the quantum system layout. For instance, as aforementioned, it is reasonable to assume that the execution of a PQC on `ibmq_cusco` involves no more than three gate errors. Extreme scenarios with higher numbers of gate errors can be disregarded, as large fluctuations with very low probabilities are effectively averaged out during execution shots without introducing significant bias.

IV. Methodology: Equitable Training with Low-Complexity Search

A. Problem Formulation

We aim to co-optimize the QNN accuracy under general noises and fatal error cases, given error set $\Omega_{\boldsymbol{e}}$. We specifically consider fatal accuracy by actively searching for and sampling the fatal errors during optimization. In this error-rate-independent optimization problem, we do not rely on specific error rates; therefore, we treat the first component as an error-free term. The optimization problem is formulated as follows:

$$\min_{\boldsymbol{\theta}} \quad \mathcal{L}(f(|\mathbf{X}\rangle, \boldsymbol{\theta}), \mathbf{Y}) + \lambda \mathcal{L}_{sup}(\Omega_{\boldsymbol{e}}, \mathbf{X}, \mathbf{Y}) \tag{4}$$

where $(\mathbf{X}, \mathbf{Y})$ is the input-label pairs following distribution in sample space $\mathcal{X} \times \mathcal{Y}$, and $\lambda > 0$ is a hyperparameter to balance the focus on error-free accuracy and fatal accuracy.

TABLE I: Estimated $\mathcal{L}_{sup}(\Omega_e)$ on a data batch and time consumption on different strategies. The first column is the QNN scale and the maximal expected gate errors in Ω_e, i.e., (G, M). Search time is measured on Intel i7 @3.80GHz CPU.

Search Size	BruteForce		RandomSearch		**Low-Complexity Search**	
	$\mathcal{L}_{sup}(\Omega_e)$	Time(s)	$\widehat{\mathcal{L}}_{sup}(\Omega_e)$	Time(s)	$\widehat{\mathcal{L}}_{sup}(\Omega_e)$	Time(s)
$(24, 2)$	1.19	30	1.07	1.73	1.15	1.22
$(40, 2)$	1.11	122	1.03	4.24	1.06	1.74
$(56, 2)$	1.22	317	1.06	7.37	1.22	2.26
$(24, 3)$	1.31	655	1.12	2.57	1.27	1.20
$(40, 3)$	1.24	4,617	0.97	6.24	1.11	1.75
$(56, 3)$	1.23	15,964	1.09	10.40	1.22	2.35

B. Low-Complexity Search

To determine the supremum of losses, one can naively traverse all error cases in Ω_e. However, the cardinality of the error set $|\Omega_e|$ actually increases polynomially along the PQC scale, i.e., the gate number G. Since $\mathcal{L}_{sup}$ must be evaluated for each data batch and at every iteration during training, the computational time quickly becomes impractical as the PQC scales up. Therefore, it is essential to adopt low-complexity yet effective search strategies to address this scalability challenge.

Scalable Evolution-Based Low-Complexity Search. We adopt the idea from genetic algorithm [20] to search for fatal errors during each training iteration. To balance the search effort and effectiveness, we propose allowing the error cases to evolve alongside the model optimization. Under this strategy, the search complexity in each iteration only depends on the error population N_{pop} in current generation.

Low-Complexity Search Validation in Fatal Loss Estimation. The evaluation of $\mathcal{L}_{sup}(\Omega_e)$ must be performed repeatedly during QNN training, thus the key to reducing time consumption is to use fewer samples to derive a highly accurate estimate. In Table I, we demonstrate the search procedure between brute-force search, random search, and our low-complexity search. We construct three QNNs with scales ($G = 24, 40, 56$) on the MNIST-2 task, and vary Ω_e by collecting all cases up to two ($M = 2$) and three ($M = 3$) errors. While brute-force search explores all error cases in Ω_e, it guarantees an accurate fatal loss estimate, but at the cost of significant computational effort. Random search (utilized in [7]), though less exhaustive, fails to provide sufficiently accurate estimates of fatal loss. Our evolution-based low-complexity search strategy offers comparable estimates to sequential search but with approximately half the time cost for small-to-moderate QNN scales; this speedup even increases over 4 times on large QNN. Moreover, the time cost of our low-complexity search grows only marginally with the number of errors, e.g., only around 4% from two errors to three errors, while this increment is significant on the other two strategies.

C. Overview of Equitable Training

To solve the optimization problem presented in Eq. 4, we iteratively update the parameters $\boldsymbol{\theta}$ ithrough equitable training, as detailed in Algorithm 1. In each iteration, we perform signle-generation low-complexity search to estimate the supremum of $\mathcal{L}_{sup}(\Omega_e)$ for each data batch, so that error candidates evolve with the QNN training. This is followed by evaluating both the error-free model and the noisy model with identified fatal error $\boldsymbol{E}^\star$. Crucially, we conduct multiple searches per iteration to frequently refresh the candidate set $\mathcal{S}_e$. This approach is essential for enhancing the robustness of the QNN model against critical errors, given the large size of Ω_e and the necessity of covering fatal error cases for effective mitigation.

Algorithm 1 Equitable noise-aware training on QNN

Require: error case set Ω_e, QNN model f, training epoch T, scaling factor λ, learning rate η
Ensure: $\boldsymbol{\theta}^\star = \boldsymbol{\theta}^T$ for equitable QNN

```
initialize θ^0 and candidates S_e ⊂ Ω_e;
for t = 1..T do
    for data batch x do
        E*, {L_E} ← sup_{E∈S_e} L(f(|x⟩, θ^{t-1}|E), y)
                                  ▷ Deterministic noise model
        S_e ⊂ Ω_e ← crossover&mutation(S_e, {L_E})
        ∂L/∂θ, ∂L'/∂θ ← ∂L(f(|x⟩,θ^{t-1}),y)/∂θ^{t-1}, ∂L(f(|x⟩,θ^{t-1}|E*),y)/∂θ^{t-1}
        θ^t ← θ^{t-1} − η ∂L/∂θ − ηλ ∂L'/∂θ
    end for
end for
```

λ **Selection.** λ provides the trade-off between the normal accuracy and fatal accuracy of the QNN models. Although the determination of λ can be evaluated by hyperparameter tuning, we share an empirical start as $\lambda = 0.5$.

V. Evaluation

Benchmark Strategies. All QNN tasks are evaluated under error-free training (baseline) and noise injection training (NI) strategies, as benchmark methods. For baseline, we optimize QNN models excluding noise consideration during QNN training. For NI, we follow the assumption in [7] that one gate has the same static rates to induce `Pauli-X`, `Pauli-Y`, and `Pauli-Z` error. For the injecting error rate, we generally categorize the error rates of a QNN model as "NI-L(ight)" ($p^X = p^Y = p^Z = 0.001$ for all qubits), "NI-M(oderate)" ($p = 0.005$), and "NI-H(eavy)" ($p = 0.01$).

Metric. We apply fatal accuracy (F.Acc) and noisy accuracy (N.Acc) to evaluate different NAT strategies. Specifically, fatal accuracy is derived from the lowest accuracy from error cases of Ω_e that contain high-probability errors. On the other hand, the noisy accuracy measures the model performance under a certain set of error rates on qubits. As the qubit error rate is unpredictable and dispersed (Section III)[1], we randomly pick three noisy environments based on the error rate levels. We denote $p_g = p_g^X = p_g^Y = p_g^Z$ on g-th gate, and each gate's error rate follows Gaussian distribution $p_g \sim \mathcal{N}(\mu_p, \sigma_p^2)$ s.t. $p_g > 0$ for a G-gate PQC. N.Acc(L) – the noisy accuracy under low error rates, $\mu_p = 0.001$ and $\sigma_p = 0.0005$; N.Acc(M) – for medium error rates $\mu_p = 0.005$ and $\sigma_p = 0.002$; N.Acc(H) – for high error rates $\mu_p = 0.01$ and $\sigma_p = 0.005$.

[1]This is why we avoid using real quantum environments, as their error rates cannot be accurately quantified or customized.

TABLE II: The fatal accuracy (F.Acc) on QNN models (with 3/5/7/10 layers), trained by the baseline, noise-injection, and ours strategy. The evaluation tasks are MNIST (M2, M4, M10), FMNIST(F2, F4), and CIFAR(C2). All the models are trained on 50 epochs, with the same training configuration. Best F.Acc for each task is **highlighted**. Additionally, improvements of ours over the second-best are marked in green, while the difference to the best is in red when ours is not the best.

	3Layer						5Layer				7Layer				10Layer			
	M2	M4	M10	F2	F4	C2	M2	M10	F2	F4	M2	M10	F2	C2	M2	F2	F4	C2
Baseline	0.303	0.023	0.067	0.247	0.099	0.380	0.152	0.025	0.163	0.059	0.146	0.011	0.166	0.378	0.141	**0.195**	0.056	0.341
NI-L	0.378	0.047	0.094	0.293	0.141	0.453	0.152	0.025	0.165	0.055	0.136	0.027	0.152	0.387	0.135	0.160	0.055	0.396
NI-M	0.273	0.033	0.062	0.333	0.147	0.444	0.148	0.023	0.164	0.041	0.149	0.030	0.167	0.388	0.127	0.164	0.039	0.367
NI-H	0.390	0.059	0.100	0.498	0.108	0.501	0.179	0.042	0.184	0.044	0.152	**0.089**	0.164	0.368	0.174	0.171	0.052	**0.397**
ours	**0.564**	**0.136**	**0.139**	**0.673**	**0.272**	**0.536**	**0.184**	**0.074**	**0.211**	**0.064**	**0.191**	0.086	**0.198**	**0.407**	**0.183**	0.173	**0.057**	0.390
(+/−)	+0.174	+0.077	+0.039	+0.175	+0.126	+0.036	+0.004	+0.032	+0.027	+0.005	+0.039	-0.003	+0.031	+0.019	+0.009	-0.022	+0.001	-0.007

A. *Problem Solving: Image Classification*

As image classification on QNN is still emerging, common QNN models can achieve good performance on easy datasets, such as MNIST [21], FMNIST [22], and CIFAR [23]. We evaluate the tasks MNIST2, MNIST4, MNIST10, FMNIST2, FMNIST4, and CIFAR2, which all have four-qubit input except MNIST10 with ten qubits.

QNN Configuration. W.L.O.G, we build the QNN with `U3` gates and `CU3` (Controlled-U3) gates. We define one layer as "`U3+CU3`" layer, where each qubit has a `U3` gate followed by a `CU3` gate cyclically [12]. We specify four QNN models, with 3 layers, 5 layers, 7 layers, and 10 layers. To apply our equitable training, we empirically assume $M = 1, 2, 2,$ and 3 for each QNN model, representing larger PQC scales with more expected gate errors in practice.

Fatal Accuracy Evaluation. We demonstrate the fatal accuracies of different training strategies on our selected QNN models in Table II. From an overall standpoint, our equitable training strategy consistently achieves the highest fatal accuracy across most tasks. It only slightly underperforms compared to the top-performing strategies on some large-scale tasks, such as the 7-layer MNIST10, where the difference is minimal. By comparing the NI strategy with the baseline method, we only observe a marked improvement in fatal accuracy in smaller QNNs (e.g., 3 layers). However, the advantage of the NI strategy diminishes for moderate to large QNNs (5 layers and above). This is evidenced by the NI strategy's marginal performance improvement over the baseline in fatal accuracy assessments. Only the "NI-Heavy" method, which has a higher likelihood of sampling errors, occasionally outperforms the baseline method with certain improvements.

On the other hand, our equitable training strategy significantly outperforms the NI strategy in small-scale QNNs, e.g., achieving a 17.51% improvement in FMNIST2 for 3-layer QNN. Due to more frequent access of fatal errors during training, our strategy also excels in moderate-scale QNN models, such as those with 5 layers. Yet, as the model scale increases (in 10-layer QNNs), our advantage diminishes, indicating reduced access to fatal errors and less optimization of the PQC parameters regarding these errors. Despite these challenges, our strategy still manages to deliver the best performance in over half of the large-scale QNN tasks.

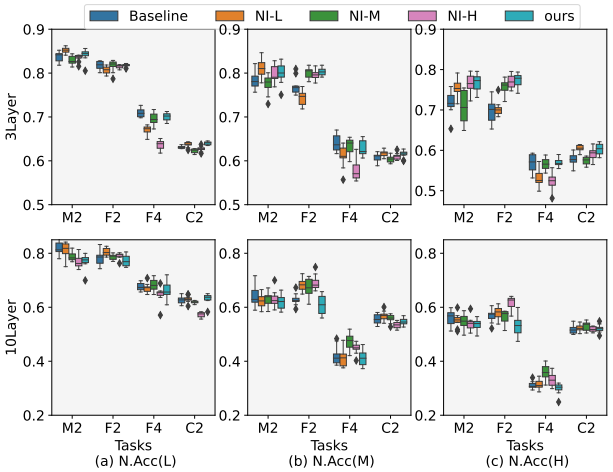


Fig. 4: The noisy accuracy (N.Acc) of the 3/10-layer QNN model, optimized using various strategies under (L)ow, (M)edium, and (H)igh error rates.

Noisy Accuracy Evaluation. In Fig. 4, we demonstrate the noisy accuracy of the small (3-layer) and large (10-layer) QNN models in different noisy environments. By incorporating additional parameters into PQC, the 10-layer model exhibits a higher likelihood of errors in noisy environments than 3 layers, leading to an obvious decline in noisy accuracy. Notably, additional layers would typically enhance performance without noise, creating antagonsim with the detrimental effects of more errors in the noisy environment. This is demonstrated by the evaluation for CIFAR2 task across 3-layer to 10-layer QNN, where the 10-layer QNN surpasses the 3-layer model under low error conditions (N.Acc(L)), yet accuracy decreases with further layer additions. In most cases, the adverse impact of errors outweighs the benefits of increased model complexity. Therefore, effective error mitigation strategies are essential for practical QML tasks.

Analysis of different strategies reveals that the baseline method excels only in simpler tasks, such as MNIST2, while strategies that incorporate noise considerations generally outperform the baseline in noisy environments across other tasks. However, this advantage is not markedly pronounced. Moreover, when compared to NI strategies, our equitable training achieves comparable noisy accuracy, leading in 8 out of 24 tasks.

TABLE III: The N.Acc of 3-layer QNN under different noise mitigation strategies, under noisy environment of $p = 0.01$. Accuracies are collected from 10 runs. EX.—Extrapolation.

	M2	M4	F2	F4
Baseline	$0.718^{\pm 0.015}$	$0.488^{\pm 0.023}$	$0.710^{\pm 0.014}$	$0.576^{\pm 0.010}$
ours	$0.763^{\pm 0.009}$	$0.520^{\pm 0.014}$	$0.774^{\pm 0.007}$	$0.562^{\pm 0.012}$
ours+EX.	$0.811^{\pm 0.015}$	$0.639^{\pm 0.008}$	$0.802^{\pm 0.007}$	$0.642^{\pm 0.010}$
ours+Norm.	$0.766^{\pm 0.026}$	$0.567^{\pm 0.022}$	$0.772^{\pm 0.032}$	$0.616^{\pm 0.022}$
error-free	0.856	0.691	0.822	0.715

TABLE IV: Fatal accuracy (F.Acc) and noisy accuracy(N.Acc) for the POS tagging task under different NAT strategies. The noisy accuracy is averaged on 10 runs.

	F.Acc	N.Acc(L)	N.Acc(M)	N.Acc(H)
Baseline	0.35	$0.92^{\pm 0.02}$	$0.90^{\pm 0.02}$	$0.87^{\pm 0.03}$
NI-L	0.18	$0.90^{\pm 0.01}$	$0.87^{\pm 0.03}$	$0.85^{\pm 0.04}$
NI-M	0.20	$0.94^{\pm 0.01}$	$0.92^{\pm 0.02}$	$0.89^{\pm 0.04}$
NI-H	0.13	$0.89^{\pm 0.01}$	$0.88^{\pm 0.03}$	$0.85^{\pm 0.05}$
Ours	**0.69**	$\mathbf{0.94}^{\pm 0.01}$	$\mathbf{0.93}^{\pm 0.01}$	$\mathbf{0.90}^{\pm 0.04}$

Combined with our method's superior performance in fatal accuracy, our equitable training not only matches the general performance of SOTA NAT strategies without compromise, but also significantly enhances the robustness of the QNN models against fatal errors during practical execution.

NAT with Noise Mitigation. We assess the integration of our NAT strategy with post-training noise mitigation techniques in Table III. For the extrapolation method [24], we selecte 10 random error rates between 0.001 and 0.05 during QNN inference and extrapolate to the error-free expectation value of the PQC measurement. In measurement normalization [7], the QNN model is trained to normalize each qubit measurement across data batches. As our training strategy imporves N.Acc on most tasks compared to the baseline, post-training noise mitigation further improves inference accuracy, approximating error-free conditions. Among the strategies, extrapolation yield more substantial improvements due to multiple measurements, whereas measurement normalization provides enhancements with a single inference run.

B. Problem Solving: POS Tagging

We take a case study of the part-of-speech (POS) tagging task to evaluate the NAT strategies in another QNN architecture, i.e., quantum long short-term memory (QLSTM) [25]. Given that current QLSTM models can only be performed on small-scale datasets [26], popular ones (such as Penn Treebank [27]) are not practical for QLSTM application, and to our knowledge there is no corresponding work on them. Thus, we generate a small corpus that contains 20 sentences, whose vocabulary tags contain noun, verb, determiner, adjective, adverb, and pronoun.

Results. In Table IV, we evaluate the fatal accuracy and noisy accuracy on QNN models optimized with different strategies. For the fatal accuracy, our training strategy can achieve a significant improvement over baseline and noise-injection methods. Since the QNN scale and the cardinality of Ω_e are small and we have enough search times during the training iteration, the fatal loss is well addressed. Besides, since the QLSTM architecture is a hybrid model, i.e., classical NN plus QNN, the parameters in the classical part can also help the robustness of QNN against fatal errors. In addition, our method can achieve comparable noisy accuracies with the noise-injection strategy under various error rates. This observation again proves the superiority of our method to provide robustness in fatal errors and dynamic noise environments.

TABLE V: The fatal MSE loss (F.Loss) and noisy MSE loss (N.Loss) with different error rates, for the regression task under different NAT strategies. The noisy loss is averaged on 10 runs.

	F.Loss	N.Loss(L)	N.Loss(M)	N.Loss(H)
Baseline	1.958	$\mathbf{0.008}^{\pm 0.025}$	$0.075^{\pm 0.102}$	$0.404^{\pm 0.279}$
NI-L	1.261	$0.029^{\pm 0.058}$	$0.076^{\pm 0.195}$	$0.245^{\pm 0.156}$
NI-M	1.695	$0.055^{\pm 0.096}$	$0.059^{\pm 0.073}$	$0.503^{\pm 0.249}$
NI-H	1.292	$0.010^{\pm 0.026}$	$\mathbf{0.054}^{\pm 0.070}$	$0.196^{\pm 0.172}$
Ours	**0.543**	$0.048^{\pm 0.019}$	$0.123^{\pm 0.083}$	$\mathbf{0.134}^{\pm 0.088}$

C. Problem Solving: Regression

QNNs have been explored in regression tasks [28]. We build the QNN as a 3-layer `U3+CU3` model with 2 qubits, which has in total $G = 12$ gates, thus the Ω_e collects all the error cases where $M = 1$. We evaluate the regression task to fit the mapping function: $y = \sin(2x_1)\cos(x_2)$.

Results. Table V shows the fatal and noisy losses on training strategies evaluated. Our equitable training can achieve at least 57% reduction on the fatal loss over other strategies. For noisy loss, the divergence of performance of different methods is significant. The baseline strategy without error consideration is prone to have lower loss in low error rate environment, while the noise injection with high-error-rate (NI-H) and our method perform better in high error rate environment. In addition, we notice that the noisy loss evaluation is not accurate on large error rate, w.r.t. large standard deviation to the average N.Loss. This is also a motivation that we propose the F.Loss metric to evaluate the robustness of a model against error, which is error-rate independent.

VI. Conclusion

We highlight the critical impact of dynamic noise and fatal errors on QNNs. Existing training and evaluation methods fall short facing a dynamically changing quantum noise environment. We propose fatal error analysis, which is independent of error rates and assesses QNN robustness against high-probability fatal errors. We also present the first noise-aware training strategy aimed at general fatal error mitigation. Our evaluations confirm the efficacy of our strategy for fatal errors; besides, QNNs optimized by our approach perform on par with SOTA NAT strategies under various error rates.

Acknowledgment

Shijin Duan and Xiaolin Xu were supported in part by the US NSF under CNS-2326597, CNS-2239672, and a Cisco Research Award. Shaolei Ren was supported in part by the US NSF under CNS-2326598.

REFERENCES

[1] M. Sajjan, J. Li, R. Selvarajan, S. H. Sureshbabu, S. S. Kale, R. Gupta, V. Singh, and S. Kais, "Quantum machine learning for chemistry and physics," *Chemical Society Reviews*, 2022.

[2] W. Guan, G. Perdue, A. Pesah, M. Schuld, K. Terashi, S. Vallecorsa, and J.-R. Vlimant, "Quantum machine learning in high energy physics," *Machine Learning: Science and Technology*, vol. 2, no. 1, p. 011003, 2021.

[3] R. Guarasci, G. De Pietro, and M. Esposito, "Quantum natural language processing: Challenges and opportunities," *Applied Sciences*, vol. 12, no. 11, p. 5651, 2022.

[4] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio *et al.*, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.

[5] O. Simeone *et al.*, "An introduction to quantum machine learning for engineers," *Foundations and Trends® in Signal Processing*, vol. 16, no. 1-2, pp. 1–223, 2022.

[6] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[7] H. Wang, J. Gu, Y. Ding, Z. Li, F. T. Chong, D. Z. Pan, and S. Han, "Quantumnat: quantum noise-aware training with noise injection, quantization and normalization," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1–6.

[8] Z. Cai, R. Babbush, S. C. Benjamin, S. Endo, W. J. Huggins, Y. Li, J. R. McClean, and T. E. O'Brien, "Quantum error mitigation," *Reviews of Modern Physics*, vol. 95, no. 4, p. 045005, 2023.

[9] S. Endo, Z. Cai, S. C. Benjamin, and X. Yuan, "Hybrid quantum-classical algorithms and quantum error mitigation," *Journal of the Physical Society of Japan*, vol. 90, no. 3, p. 032001, 2021.

[10] A. Strikis, D. Qin, Y. Chen, S. C. Benjamin, and Y. Li, "Learning-based quantum error mitigation," *PRX Quantum*, vol. 2, no. 4, p. 040330, 2021.

[11] G. S. Ravi, K. Smith, J. M. Baker, T. Kannan, N. Earnest, A. Javadi-Abhari, H. Hoffmann, and F. T. Chong, "Navigating the dynamic noise landscape of variational quantum algorithms with qismet," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 515–529.

[12] H. Wang, Y. Ding, J. Gu, Y. Lin, D. Z. Pan, F. T. Chong, and S. Han, "Quantumnas: Noise-adaptive search for robust quantum circuits," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 692–708.

[13] F. Yan, A. M. Iliyasu, and S. E. Venegas-Andraca, "A survey of quantum image representations," *Quantum Information Processing*, vol. 15, pp. 1–35, 2016.

[14] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[15] K. Khodjasteh and L. Viola, "Dynamically error-corrected gates for universal quantum computation," *Physical review letters*, vol. 102, no. 8, p. 080501, 2009.

[16] D. Greenbaum and Z. Dutton, "Modeling coherent errors in quantum error correction," *Quantum Science and Technology*, vol. 3, no. 1, p. 015007, 2017.

[17] R. Qin, Z. Liang, J. Cheng, P. Kogge, and Y. Shi, "Improving quantum classifier performance in nisq computers by voting strategy from ensemble learning," *arXiv preprint arXiv:2210.01656*, 2022.

[18] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth *et al.*, "The variational quantum eigensolver: a review of methods and best practices," *Physics Reports*, vol. 986, pp. 1–128, 2022.

[19] R. Acharya *et al.*, "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, no. 7949, pp. 676–681, 2023.

[20] S. Mirjalili and S. Mirjalili, "Genetic algorithm," *Evolutionary algorithms and neural networks: theory and applications*, pp. 43–55, 2019.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[23] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[24] K. Temme, S. Bravyi, and J. M. Gambetta, "Error mitigation for short-depth quantum circuits," *Physical review letters*, vol. 119, no. 18, p. 180509, 2017.

[25] S. Y.-C. Chen, S. Yoo, and Y.-L. L. Fang, "Quantum long short-term memory," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 8622–8626.

[26] R. Di Sipio, J.-H. Huang, S. Y.-C. Chen, S. Mangini, and M. Worring, "The dawn of quantum natural language processing," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 8612–8616.

[27] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," 1993.

[28] G. Wang, "Quantum algorithm for linear regression," *Physical review A*, vol. 96, no. 1, p. 012335, 2017.

Assessing Quantum Layout Synthesis Tools via Known Optimal-SWAP Cost Benchmarks

Shuohao Ping[1], Wan-Hsuan Lin[1], Daniel Bochen Tan[2], Jason Cong[1]
[1]Computer Science Department, University of California, Los Angeles, CA, USA,
[2]Harvard University Cambridge, MA, USA
{sp1831, wanhsuanlin}@ucla.edu; danieltan@g.harvard.edu; cong@cs.ucla.edu

***Abstract*—Quantum layout synthesis (QLS) is a critical step in quantum program compilation for superconducting quantum computers, involving the insertion of SWAP gates to satisfy hardware connectivity constraints. While previous works have introduced SWAP-free benchmarks with known-optimal depths for evaluating QLS tools, these benchmarks overlook SWAP count—a key performance metric. Real-world applications often require SWAP gates, making SWAP-free benchmarks insufficient for fully assessing QLS tool performance. To address this limitation, we introduce QUBIKOS, a benchmark set with provable-optimal SWAP counts and non-trivial circuit structures. For the first time, we are able to quantify the optimality gaps of SWAP gate usages of the leading QLS algorithms, which are surprisingly large: LightSabre from IBM delivers the best performance with an optimality gap of 63x, followed by ML-QLS with an optimality gap of 117x. Similarly, QMAP and t$|$ket$\rangle$ exhibit significantly larger gaps of 250x and 330x, respectively. This highlights the need for further advancements in QLS methodologies. Beyond evaluation, QUBIKOS offers valuable insights for guiding the development of future QLS tools, as demonstrated through an analysis of a suboptimal case in LightSABRE. This underscores QUBIKOS's utility as both an evaluation framework and a tool for advancing QLS research.**

I. Introduction

Superconducting qubit has emerged as a promising solution for realizing large scale quantum computing. However, the limited qubit connectivity in superconducting quantum computers restricts two-qubit gates to specific pairs of qubits. In order to transform quantum circuits into hardware-executable forms, we need quantum layout synthesis (QLS) to insert SWAP gates that enable arbitrary two-qubit gates in the circuit. However, these SWAP gates increase circuit size and depth, reducing overall fidelity. Thus, minimizing the number of inserted SWAP gates is a primary objective of QLS.

The QLS problem is proved to be NP-hard [1]. The existing methods fall into two classes: exact approaches or heuristic algorithms. The exact approaches formulate a QLS problem into a constraint satisfaction problem and can obtain optimal results [2]–[10] but it cannot scale up to solve problems with a hundred qubits. To address scalability issues, several heuristic algorithms are proposed [1], [11]–[27]. These methods are efficient, yet exhibit performance degradation as the problem size increases. While there is a clear runtime advantage of heuristic algorithms over exact methods, the trade-off in solution quality remains uncertain due to the lack of benchmarks with known optimal SWAP counts. Measuring the optimality gap using exact tools is an alternative, but this approach becomes impractical for large circuits due to scalability issues. Therefore, having a benchmark set with known optimal SWAP counts is crucial for enabling solid comparison of QLS tools.

Few studies have attempted to establish such benchmark sets. Tan and Cong introduced the QUEKO benchmark with known-optimal depth and zero SWAP gates [28]. However, this benchmark is limited in scope; it can be addressed by QLS tools employing subgraph isomorphism and fails to evaluate tool performance on circuits that require SWAP gates. Li et al. proposed the QUEKNO benchmark [29], which generates circuits requiring SWAP gates by deriving a series of mappings from an initial mapping and selected SWAP operations. However, these circuits do not have known optimal SWAP counts, and the solutions are only claimed to be near-optimal without rigorously quantifying the optimality gap. As a result, QUEKNO cannot measure the optimality gap of QLS tools. To overcome the limitations of existing benchmarks, we propose QUBIKOS (**QU**antum **B**enchmark w**I**th **K**nown **O**ptimal **S**wap counts), which is the first benchmark set with a known optimal non-zero SWAP count. It permits a more comprehensive assesment of QLS tools and supports the development of future advancements in this area.

With QUBIKOS, we evaluate four QLS tools on four architectures and observe the optimality gaps. We find that as the size of the architecture grows, the optimality gap also grows from 1x to 233.97x even for the best performing tools. This shows an urgent need to have a more effective QLS tool to fully utilize the power of large systems. Furthermore, for architectures with a similar size, the structure of the architecture also affects the optimality gap. For instance, the optimality gap for IBM Rochester is seven times larger than Google Sycamore as a result of sparse connectivity and fewer axes of symmetry.

QUBIKOS provides a qualitative evaluation on the performance of QLS tools by measuring the optimality gap between QLS results and optimal results. We can control the circuit size and number of optimal SWAP gates in the QUBIKOS circuit, which allows us to test the QLS tools under different conditions. QUBIKOS also offers insights on improving the performance of QLS tools by analyzing examples when QLS tools fails to find optimal SWAP gate counts.

The rest of this paper is organized as follows. Section II introduces the quantum layout synthesis problem. Section III details the construction of QUBIKOS benchmark. Section IV evaluates existing QLS tools with QUBIKOS, and Section V concludes the paper.

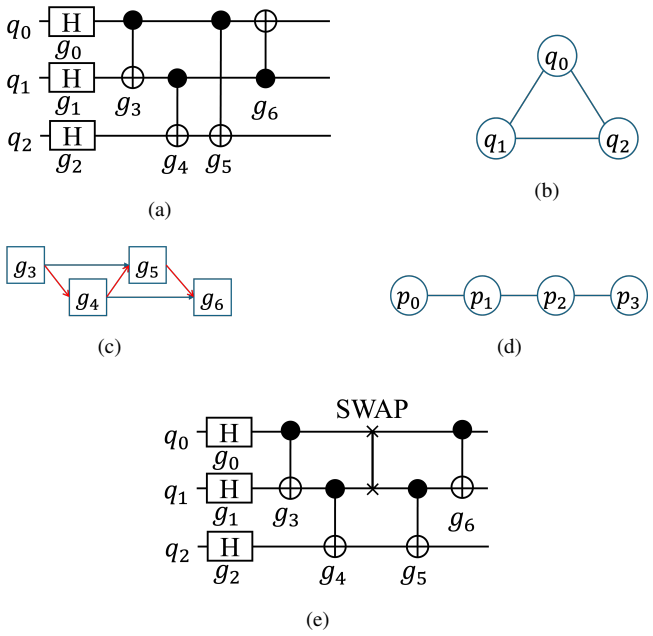


Figure 1. (a) An example of a quantum circuit. (b) The interaction graph for the circuit shown in (a). (c) Gate dependency graph derived from the circuit. The red edges mark a dependence chain from g_3 to g_5.(d) A device coupling graph with 4 physical qubits in a line architecture. (b) A transpiled circuit of (a) with one inserted SWAP gate for the line architecture with mapping $q_0 \to p_0, q_1 \to p_1$, and $q_2 \to p_2$.

II. Quantum Layout Synthesis

The QLS problem arises from the need to map gates in a quantum circuit to a superconducting quantum processor characterized by a coupling graph and perform necessary transformations to the circuit via inserting SWAP gates. The inputs to the QLS problem are: *quantum circuit* and *coupling graph*.

A *quantum circuit* is defined as a sequence of single and two-qubit gates. In this paper, we denote the set of program qubits by Q, a program qubit by $q \in Q$, the set of single-qubit gates by $\mathcal{G}_1$, the set of two-qubit gates by $\mathcal{G}_2$, and the overall gate set by $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$. In addition, for a two-qubit gate $g(q, q') \in \mathcal{G}_2$, we use $g[0] = q$ and $g[1] = q'$ to denote the two program qubits g operates on. Figure 1(a) is an example of a quantum circuit where H gates are single qubit gates, and CNOT gates are two-qubit gates.

A *coupling graph* $G_C(P, E_P)$ describes the connectivity of physical qubits for a given hardware where P is a set of physical qubits and E_P is a set of edges, where two-qubit gates can be performed. Figure 1(d) depicts a coupling graph. We define the function $Neighbor(v, G)$ to be the set of adjacent vertices of v in G. For example, the adjacent physical qubits of a physical qubit $p \in P$ is denoted as $Neighbor(p, G_C)$,

According to a quantum circuit C, we define the *interaction graph* $G_I(Q, E_Q)$, where each program qubit is a node and an edge exists between q and q' if $g(q, q') \in \mathcal{G}_2$. Figure 1(b) demonstrates the interaction graph of the circuit in Figure 1(a). The neighbor of a program qubit $q \in Q$ in the interaction graph $G_I(Q, E_Q)$ is designated by $Neighbor(q, G_I)$.

The *gate dependency graph* $D(\mathcal{G}_2, E_G)$ is a directed acyclic graph that represents the execution order of two-qubit gates. Single-qubit gates are excluded from D since they do not impose connectivity constraints, and can be inserted back to the circuit after QLS. Nodes in D represent two-qubit gates in the circuit and an edge (g, g') is drawn if g' can be executed immediately after g. Figure 1(c) shows an example of D for the circuit in Figure 1(a). There is an edge from g_3 to g_4 because they both act on q_1, and g_4 comes after g_3. In general, for any given gates $g, g' \in G_2$, if g' must execute before g, there exists a path from g' to g in D. We define the set of previous gates of gate g as $Prev(g) = \{g' | \exists \text{path from } g' \text{ to } g \text{ in } D\}$.

The output of the QLS problem is a *transpiled circuit* with inserted SWAP gates, and an initial mapping from program qubits to physical qubits $f : Q \to P$. Figure 1(e) illustrates a valid QLS result using one SWAP to map the circuit in Figure 1(a) to the coupling graph in Figure 1(d). A transpiled circuit can be written in the form of $C_0 \cdot T_0 \cdot C_1 \cdot T_1 \cdot ... \cdot T_{n-1} \cdot C_n$, where C_i is a partial gate sequence with $C_0 \cdot C_1 \cdot ... \cdot C_n = C$, and T_i is a SWAP gate that implements mapping transformation.

III. QUBIKOS Circuit Construction

To construct a circuit with an optimal SWAP count of n given a coupling graph G_C, the first step is to generate n circuit segments. Each requires exactly one SWAP gate under all possible mappings, which means the interaction graph is not isomorphic to any subgraph of G_C, as such isomorphisms would allow execution without SWAP insertion. Figure 2(c) is a non-isomorphic interaction graph to the coupling graph G_C displayed in Figure 2(a) as the interaction graph contains a node with degree 5, while the maximum degree in G_C is 4.

However, simply concatenating these graphs does not guarantee an optimal SWAP count. Thus, the second step is to construct gate dependencies both within and between the interaction graphs and enforce that gates from different interaction graphs execute in serial. This guarantees that when we concatenate the segments to construct the final circuit, the corresponding optimal SWAP count equals the sum of the optimal SWAP counts for each segment.

Figure 3 shows a gate dependency graph of a circuit that requires 2 SWAP gates. Here, g_4 and g_{15} partition the circuit into two sets: $g_0 - g_4$ and $g_5 - g_{15}$, with no gates from different sets executed in parallel. These sets form two non-isomorphic interaction graphs that must execute serially, ensuring each requires one SWAP gate to resolve connectivity, and no single SWAP can resolve conflicts in both graphs.

Repeating this process n times, we can generate a circuit C with n SWAP gates. Finally, we can insert additional gates that do not increase the optimal SWAP count, allowing for arbitrary circuit structures and making it more challenging to solve. The following sections provide detailed examples of the subroutines.

A. Non-isomorphic Interaction Graph Generation

The first step constructs non-isomorphic interaction graphs relative to a given architecture G_C. Starting with a random

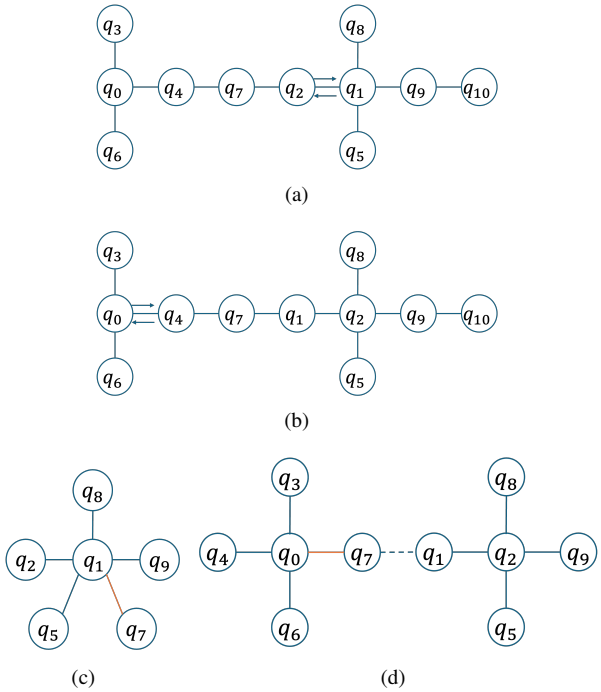


Figure 2. (a) The device coupling graph G_C with a random initial mapping. Arrows represent the first SWAP operation. (b) The device coupling graph with qubit mapping after SWAP(q_1, q_2) on the initial mapping. Arrows indicate the second SWAP operation. (c) The interaction graph $G_{I_1}(Q, E_{Q_1})$ that enforces SWAP(q_0, q_4). The orange edge is the special gate. (d) The interaction graph $G'_{I_2}(Q, E'_{Q_2})$ that only considers gates after the first SWAP. The orange edge is the special gate in E_{Q_2}. The dotted line is the inserted gate. Without it, the coupling graph is G_{I_2}.

initial mapping, we select an adjacent qubit pairs for SWAP gate. The SWAP is designed to enable at least one qubit in the pair to interact with a new neighbor, ensuring the SWAP is essential. If no new interactions are introduced, the SWAP is redundant. Such edges always exist unless the coupling graph is fully connected. Figure 2(a) illustrates an example of the initial mapping on the coupling graph with SWAP gates on (q_1, q_2), enabling new interactions. For selecting subsequent SWAP gates, we need to transform the mapping based on the selected SWAP and choose an adjacent qubit pair in the new mapping that satisfies the condition discussed above. Figure 2 shows the new mapping after SWAP(q_1, q_2), and we choose edge (q_0, q_4) for the next SWAP.

To enforce a SWAP between q_1 and q_2, we construct an interaction graph based on q_1. First, q_1 must interact with all its current neighbors, ensuring it has no free edges to form new interactions. This requires a two-qubit gate on $g(q_1, q_2)$, $g(q_1, q_5)$, $g(q_1, q_8)$, and $g(q_1, q_9)$. Next, we identify a qubit that is not a neighbor of q_1 before the SWAP gate but becomes one afterward, e.g., q_7, and add the *special gate* $g(q_1, q_7)$ in the circuit. The resulting interaction graph is displayed in Figure 2(c), and is not isomorphic to any subgraph of G_C. Even though the circuit before the special gate does not need a SWAP gate, it forces the circuit to require one prior to the its execution. Any gate in the interaction graph could serve

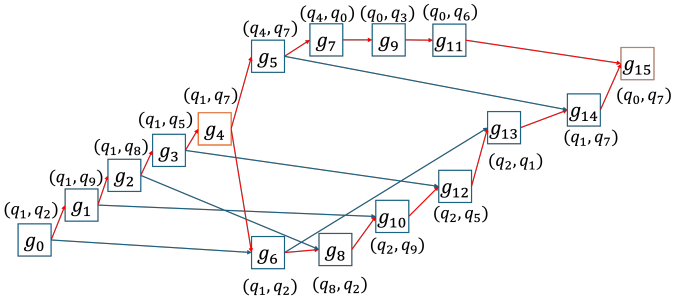


Figure 3. Gate dependency graph D for circuit C. The orange boxes represent the special gates. The red edges show the required dependence relation for g_4 and g_{15}.

as the special gate, provided the remaining four gates form an isomorphic subgraph. The special gate must execute last to ensure all other gates in the interaction graph are completed beforehand. Otherwise, it may alter the interaction graph by removing an edge and eliminate the need for a SWAP gate. Section III-B will present how to impose the dependency constraint.

Alternatively, we could use q_2 to build the interaction graph. However, simply having q_2 interact with all its neighbors plus one new neighbor qubit is insufficient. In this case, q_2 would have a degree of 3 in the interaction graph, and a mapping could exist that avoids SWAP gates by placing q_2 on a degree 3 or degree 4 node in G_C. To prevent this, all program qubits mapped to physical qubits with degrees higher than 2 must also interact with all their neighbors. This ensures all higher-degree physical nodes are occupied, making it impossible to find a new mapping that avoids SWAP gates by placing q_2 on a higher-degree physical qubit.

Next, we want to construct a circuit C to compel SWAP(q_0, q_4). After SWAP(q_1, q_2), the mapping is updated as in Figure 2(b). Using q_0 as the starting point for constructing the non-isomorphic graph, we ensure q_0 interacts with all its neighbors, generating gates $g(q_0, q_3)$ $g(q_0, q_6)$, and $g(q_0, q_4)$. Then, $g(q_0, q_7)$ serves as the special gate that enforce SWAP(q_0, q_4). In addition, q_2 must interact with all its neighbors since q_2 is mapped to a physical qubit whose degree is larger than the one that q_0 is mapped to. Figure 2(d) shows the interaction graph of G_{I_2}. Note that the dotted line is current not part of the interaction graph, but will be added in the next subroutine.

B. Gate Dependency Relation

Given the interaction graphs G_{I_1}, G_{I_2} generated by the first step, we build the QUBIKOS circuits by enforcing the gate dependency within and between interaction graphs. Considering the gates in G_{I_1}, since all these gates act on q_0, we can reorder them so that the special gate depends on all other gates, ensuring that the special gate forces a SWAP operation. For example, in the gate sequence $C_1 = [g_0(q_1, q_2), g_1(q_1, q_9), g_2(q_1, q_8), g_3(q_1, q_5), g_4(q_1, q_7)]$, the special gate $g_4(q_1, q_7)$ has to execute after all other gates.

Reordering alone does not always ensure the desired gate dependencies. For instance, in G_{I_2}, the special gate (q_0, q_7) cannot be made to execute after (q_1, q_2) simply by reordering gates in E_{Q_2} because the two disconnected components in G_{I_2} can always execute in parallel regardless of gate order. To address this, we must add edges to G_{I_2} to connect the two components. For example, we can add edge (q_1, q_7) as shown in Figure 2(d), and denote the new interaction graph as G'_{I_2}.

A correct gate order necessitates that the first special gate (q_1, q_7) is executed before all gates in E'_{Q_2}, and the second special gate (q_0, q_7) execute after the other gates in E'_{Q_2}. To satisfy the first dependency constraint, we perform a breadth-first search (BFS) on G'_{I_2}, starting from q_1 and q_7 while ignoring the edge (q_0, q_7). The order of visited edges from the BFS will establish the desired gate dependencies. In this case, the BFS result is $[(q_7, q_1), (q_0, q_4), (q_1, q_2), (q_0, q_3), (q_2, q_8), (q_0, q_6), (q_2, q_9), (q_2, q_5)]$. By prepending (q_1, q_7) to the BFS sequence, all gates will depend on it. To satisfy the second constraint, we run BFS on the same graph, starting from q_0 and q_7, reverse the order of visited edges, and append (q_0, q_7) at the end. Concatenating these two sequences yields a circuit that meets both dependency constraints. While this method does not produce the smallest possible circuit, it is valid. For simplicity, we use a smaller sequence as an example. $C_2 = [g_5(q_4, q_7), g_6(q_1, q_2), g_7(q_4, q_0), g_8(q_8, q_2), g_9(q_0, q_3), g_{10}(q_2, q_9), g_{11}(q_0, q_6), g_{12}(q_2, q_5), g_{13}(q_2, q_1), g_{14}(q_1, q_7), g_{15}(q_0, q_7)]$.

Now, we can build a circuit C with an optimal SWAP count two by concatenating C_1 and C_2, i.e., $C = C_1 \cdot C_2$. The gate dependency graph is illustrated in Figure 3, showing that gates from two interaction graphs must be executed in serial. We call C the *backbone* of the benchmark circuit and all gates between two special gates, including the later special gate, a *section* of the backbone circuit that enforce one SWAP gate.

We can introduce additional gates into the backbone without affecting the optimal SWAP gate count. For example, we can add (q_9, q_{10}) anywhere in C since it can be executed under any mapping. The key constraint is ensuring that additional gates do not introduce extra SWAP gates. For instance, (q_2, q_7) can only be inserted before g_4 since q_2 and q_7 are not neighbors after the first SWAP gate.

C. Challenges in Finding Optimal Solution for QUBIKOS

Unlike QUEKO [28], the optimal transpiled circuits for QUBIKOS benchmarks cannot be obtained using subgraph isomorphism algorithms, e.g., VF2 [30]. One may identify the special gates within the benchmark circuit by tracking the degrees of the interaction graph, and the special gates will partition the circuit into multiple sections that have an isomorphic subgraph of G. Nevertheless, running subgraph isomorphism algorithms on individual sections may not yield an optimal solution. For instance, applying VF2 to the first four gates only provides mapping information for a subset of qubits, and may not be optimal for the subsequent sections. As the optimal initial mapping for all qubits can only be determined by examining the global structure of the circuit, this benchmark is effective for testing modern QLS tools.

D. Overview for Proof of Optimality

We provide an overview for proving the optimal SWAP count in QUBIKOS circuits, but have omitted the detail proofs due to the page limit. We list the four theorems that are used to prove the optimality:

Lemma 1. *Interaction graph $G_I(Q, E_Q)$ generated by subroutine 1 forms an interaction graph that is not isomorphic to any subgraph of $G_C(P, E_P)$.*

Lemma 2. *$\forall g \in C_i$ generated by subroutine 2 has to execute after a prior special gate g_1 but before a subsequent special gate g_2. (The prior special gate for the first section is Null.)*

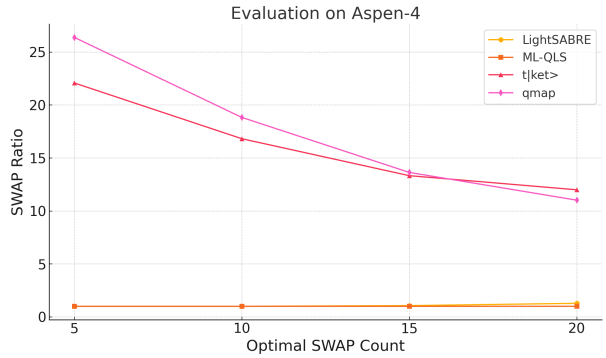
Lemma 3. $\forall 0 \leq i \leq n-1, \forall g \in C_i, \forall g' \in C_{i+1} : g \in Prev(g')$

Theorem 4. *The backbone circuit C needs at least n SWAP gates to execute.*

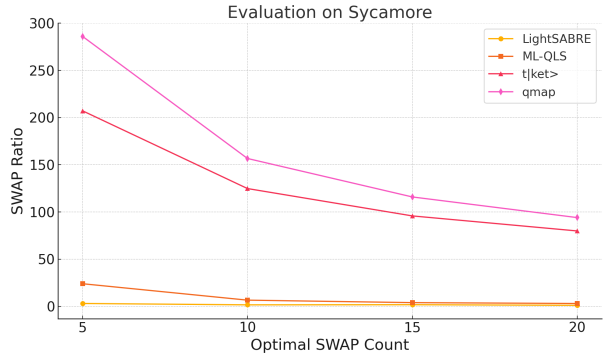
In Lemma 1, we show Section III-A always constructs a non-isomorphic interaction graph, which is trivial according to the construction. Then, we prove Lemma 2 by finding a path from g_1 to g and g to g_2 in the gate dependency graph. This is guaranteed by the BFS in Section III-B. Then, Lemma 3 states that any two gates belong to two consecutive sections of circuit C denoted as C_i and C_{i+1} must be executed in serial. This can be proved utilizing Lemma 2 by finding a path from the gate in C_i to a special gate, and a path from the special gate to the gate in C_{i+1}. Lastly, by Lemma 1 and 2, and 3, we can prove Theorem 4. Consider the backbone circuit depicted in Figure 3, we know there exists a two SWAP solution. Then, we verify that there does not exist a one SWAP solution using proof by contradiction. Assume a solution with one SWAP exists. Then, the solution will only have one SWAP gate inserted anywhere before the first special gate g_4 or after it. If the SWAP gate is inserted somewhere before g_4, then the connectivity constraints imposed by the second interaction graph G'_{I_2} remain unresolved. On the other hand, adding a SWAP gate after g_4 can't resolve the connectivity constraints in G_{I_1}. Hence, such a solution cannot exist. For a general proof, we can just extend this argument to a circuit with n sections. By Theorem 4, we need at least n SWAP gates to execute C, and we know a solution with n SWAPs. Hence, we prove the optimality.

IV. Evaluation

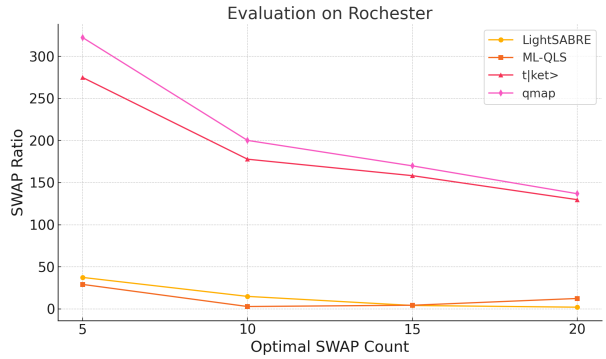
We verify the optimality of QUBIKOS in Section IV-A via OLSQ2 [10] with PySAT version 1.8.dev13 and Z3-solver version 4.13.3.0. In Section IV-B, we evaluate the optimality gap of existing QLS tool, LightSABRE from Qiskit [31] version 1.2.4, t|ket⟩ [32] version 1.34.0, QMAP [33] version 2.7.0, and ML-QLS [27]. All the experiments were conducted on Intel Xeon CPU at 2.20GHz. For LightSABRE, we test it with 1000 trials.



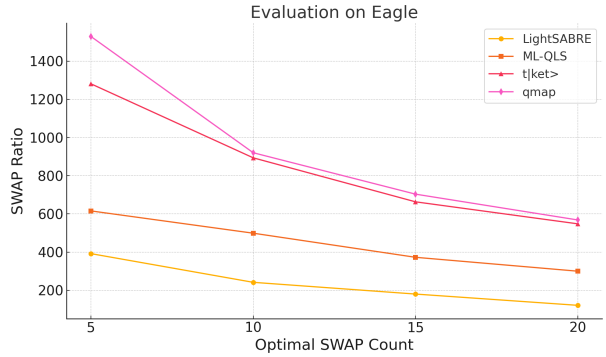
(a)



(b)



(c)



(d)

Figure 4. Results for the heuristic experiments. (a) shows the result for the Aspen-4 architecture. (b) reveals the result for the Sycamore architecture. (c) is the result for the Rochester architecture. (d) shows the result for the Eagle architecture.

A. *Optimality Study*

This experiment is to verify the optimal SWAP count for QUBIKOS circuits via the exact QLS tool OLSQ2 [10]. We generate QUBIKOS circuits for the Rigetti Aspen-4 with 16 qubits and 3x3 grid architectures. We create 400 circuits per architecture, with 100 for each SWAP count from 1 to 4. To guarantee a feasible runtime given OLSQ2's exponential complexity, we limit circuits to 30 two-qubit gates.

The optimality experiment confirms that all generated circuits have the desired SWAP counts. For each architecture, circuits designed with specific SWAP counts were verified by OLSQ2 to require the same SWAP counts, with no deviations observed. This consistency demonstrates that the benchmark circuits are optimally constructed.

B. *QLS Tool Evaluation*

This experiment is to measure the optimality gap of leading heuristic QLS tools. We produce QUBIKOS circuits for four architectures: Rigetti Aspen-4, Google Sycamore with 54 qubits, IBM Rochester with 53 qubits, and IBM Eagle with 127 qubits. For each architecture, we make 40 circuits with 10 for each SWAP count of 5, 10, 15, and 20. The two-qubit gate count is set to 300 for Aspen-4, 1500 for Sycamore and Rochester, and 3000 for Eagle. The reason for varying circuit size on different architectures is because a larger architecture requires more gates on average to construct a section of the backbone circuit as the interaction graph requiring more connections on average to be non-isomorphic.

The optimality gap is defined based on the SWAP ratio, which is calculated as the ratio of the average SWAP count to the optimal SWAP count: $\frac{Average\ SWAP\ count}{Optimal\ SWAP\ count}$. This ratio is always greater than or equal to 1, where a ratio of 1 indicates that the QLS tool finds the optimal solution.

Figure 4 presents the experimment results, showing how the optimality gap varies across different QLS tools. Among the four QLS tools, ML-QLS achieves optimal results on Aspen-4 and Rochester, and LightSABRE performs the best on the other architectures. ML-QLS performs similarly to LightSABRE, except on the Eagle architecture. This is different from the results in the ML-QLS paper [27] because we uses a newer version of Qiskit and much higher trial number. So, LightSABRE has a better performance in our experiment but not in the ML-QLS experiment. In contrast, t$|$ket$\rangle$ and QMAP exhibit a significantly larger optimality gap with an average of 185x and 207x respectively on Aspen-4 and as high as 846x and 930x on Eagle.

As the number of physical qubits increases, the optimality gap also tends to grow. This is supported by the fact that on Aspen-4, ML-QLS finds the optimal solutions in all cases. But on Sycamore, Rochester, and Eagle, LightSABRE has an optimality gap of 1.95x, 12.17x, and 233.97x respectively. However, the number of physical qubits is not the only factor; the connectivity of the coupling graph also plays a significant role. For example, despite the Rochester architecture having a similar number of physical qubits to the Sycamore architecture, its optimality gap is six times larger. This is due to

Rochester's heavy-hex structure, which has sparse connectivity and has fewer axes of symmetry compared to Sycamore's dense grid connectivity. As a result, architectures with dense connectivity are more favorable for QLS tools, leading to smaller optimality gaps.

These findings highlight the utility of our examples for evaluating QLS tools, revealing that the optimality gap varies significantly depending on the tool. While LightSABRE and ML-QLS have smaller gaps, t|ket⟩ and QMAP struggle to achieve near-optimal solutions. This emphasizes the need for more efficient QLS tools, particularly for larger architectures where the current tools demonstrate a large optimality gap. Additionally, the performance differences between the architectures suggest that the connectivity of the coupling graph influences how well QLS tools can minimize the SWAP count, with tools performing better on more connected architectures.

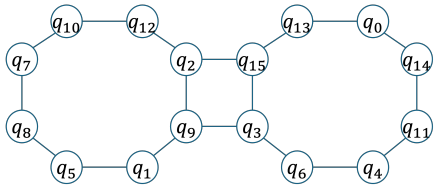
C. Case Study: LightSABRE

QUBIKOS not only provides a consistent evaluation of existing QLS tools but also offers insights for improving them. In this section, we perform a case study to analyze the results obtained from LightSABRE.

In one Aspen-4 experiment, SABRE identifies the optimal initial mapping but fails to find the correct routing, leading to a non-optimal solution. Figure 5 illustrates the partial circuit where SABRE deviates from the optimal routing. Instead of choosing to SWAP(q_3, q_9), LightSABRE opts for SWAP(q_2, q_9), resulting in additional SWAP gates and non-isomorphic mappings. To understand why LightSABRE chooses this suboptimal SWAP, we calculate the cost of the two SWAP gates. SABRE evaluates three types of costs: basic, lookahead, and decay. In this case, both SWAPs are used for the first time and connect q_2 and q_3, so the basic and decay costs are the same. The difference lies in the lookahead cost. In LightSABRE, the lookahead is controlled by the extended set, where its size is the number of gates LightSABRE will consider beyond the execution layer. In Qiskit, the extended set size is 20 with a weight of 0.5. Thus, the lookahead cost for SWAP(q_3, q_9) is 0.7, while for SWAP(q_2, q_9), it is 0.65. This occurs because the qubit distance reduction for SWAP(q_2, q_9) is greater when considering the first 20 gates equally, even though the two mappings cannot be executed under the same conditions, resulting in a non-optimal choice.

This issue could be mitigated by adding a decay factor to the lookahead cost, placing more weight on gates closer to the execution layer in D while reducing the weight for gates farther from execution. With an appropriate decay factor, SABRE could find the optimal routing.

The above analysis is specific to SABRE, but it can be done with any QLS tools to improve its performances. Moreover, this example reveals QUBIKOS circuit has a non-trivial qubit routing even when the initial mapping is optimal. This means QUBIKOS can also be utilized to evaluate standalone routers like [**?**] that require an initial mapping as input. Measuring the performance of these routers can be difficult since the quality of the initial mapping will directly impact its performance.



(a)

Gates in the extended set:	Circuit after optimal insertion:	Circuit after non-optimal insertion:
...	...	...
(q_2, q_3)	SWAP(q_3, q_9)	SWAP(q_2, q_9)
(q_3, q_2)	(q_2, q_3)	(q_2, q_3)
(q_2, q_{15})	(q_3, q_2)	(q_3, q_2)
(q_2, q_{12})	(q_2, q_{15})	SWAP(q_9, q_{15})
(q_9, q_2)	(q_2, q_{12})	(q_2, q_{15})
(q_9, q_2)	SWAP(q_3, q_2)	SWAP(q_{15}, q_{12})
(q_9, q_{15})	(q_9, q_2)	(q_2, q_{12})
(q_2, q_3)	(q_9, q_2)	SWAP(q_{12}, q_9)
(q_1, q_2)	(q_9, q_{15})	(q_9, q_2)
(q_2, q_9)	(q_2, q_3)	(q_9, q_2)
(q_2, q_3)	(q_1, q_2)	(q_9, q_{15})
(q_1, q_2)	(q_2, q_9)	(q_2, q_3)
(q_2, q_9)	(q_2, q_3)	(q_1, q_2)
(q_2, q_3)	(q_1, q_2)	(q_2, q_9)
(q_9, q_6)	(q_2, q_9)	(q_2, q_3)
(q_9, q_{15})	(q_2, q_3)	...
(q_3, q_{15})	...	
(q_3, q_{12})		
(q_{15}, q_{13})		
(q_9, q_{15})		
(q_2, q_3)		
(q_9, q_2)		
(q_1, q_2)		
...		

(b)

Figure 5. (a).Qubit mapping before LightSABRE generates non-optimal SWAP gate. (b). Portion of the circuit that LightSABRE inserts non-optimal SWAP. The left circuit is the benchmark circuit before insertion. It includes 20 gates due to the extended set size. The middle one is the circuit with optimal SWAP insertion. The right one is the non-optimal results from SABRE.

With QUBIKOS, we can assess the routers by providing the initial mapping of the generated circuit. This way, we can test the routers with the optimal initial mapping, and any non-optimal results from the routers directly relates to the design of the router itself rather than the initial mapping.

V. Conclusion

In this work, we introduce QUBIKOS, a benchmark with a known optimal SWAP count. We detail the method for generating QUBIKOS circuit and provide an overview for the proof. We verify its optimality experimentally and use it to evaluate the performance of four QLS tools. Our experiment results show that LightSABRE delivers the best performance but there is still an optimality gap that grows with the size of the architecture. Additionally, we illustrate how QUBIKOS can aid in improving QLS tools by analyzing a non-optimal case in LightSABRE, highlighting its potential to drive the development of more efficient QLS tools in the future.

VI. Acknowledgment

This research is supported in part by the National Science Foundation Award 2016245, 2313083, and the Harvard Quantum Initiative Postdoctoral Fellowship.

REFERENCES

[1] M. Y. Siraichi *et al.*, "Qubit allocation." New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 113–125.

[2] R. Wille, A. Lye, and R. Drechsler, "Optimal SWAP gate insertion for nearest neighbor quantum circuits," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2014, pp. 489–494.

[3] D. Bhattacharjee, A. A. Saki, M. Alam, A. Chattopadhyay, and S. Ghosh, "MUQUT: Multi-constraint quantum circuit mapping on NISQ computers," in *2019 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2019, pp. 1–7.

[4] R. Wille *et al.*, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[5] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *Proceedings of the 39th International Conference on Computer-Aided Design*. ACM, Nov. 2020, pp. 1–9.

[6] ——, "Optimal qubit mapping with simultaneous gate absorption," in *2021 IEEE/ACM International Conference on Computer Aided Design*. IEEE, 2021, pp. 1–8.

[7] C. Zhang *et al.*, "Time-optimal Qubit mapping." New York, NY, USA: Association for Computing Machinery, Apr. 2021, pp. 360–374.

[8] A. Molavi *et al.*, "Qubit mapping and routing via MaxSAT," in *2022 55th IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2022, pp. 1078–1091.

[9] G. Nannicini *et al.*, "Optimal qubit assignment and routing via integer programming," *ACM Transactions on Quantum Computing*, vol. 4, no. 1, pp. 1–31, 2022.

[10] W.-H. Lin *et al.*, "Scalable optimal layout synthesis for NISQ quantum processors," in *2023 60th ACM/IEEE Design Automation Conference*. IEEE, 2023, pp. 1–6.

[11] A. Ho and D. Bacon, "Announcing Cirq: an open source framework for NISQ algorithms," *Google AI Blog*, vol. 18, 2018.

[12] A. Zulehner *et al.*, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2018.

[13] IBM. (2018) Qiskit. [Online]. Available: https://qiskit.org/

[14] A. Zulehner *et al.*, "An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, Jul. 2019.

[15] M. Y. Siraichi *et al.*, "Qubit allocation as a combination of subgraph isomorphism and token swapping," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 120:1–120:29, Oct. 2019.

[16] G. Li *et al.*, "Tackling the qubit mapping problem for NISQ-era quantum devices." New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 1001–1014.

[17] P. Murali *et al.*, "Formal constraint-based compilation for noisy intermediate-scale quantum systems," *Microprocessors & Microsystems*, vol. 66, no. C, pp. 102–112, Apr. 2019.

[18] S. Sivarajah *et al.*, "t|ket⟩: a retargetable compiler for NISQ devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, Nov. 2020.

[19] A. Kole *et al.*, "Improved Mapping of Quantum Circuits to IBM QX Architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2375–2383, Oct. 2020.

[20] J. Liu *et al.*, "Not all SWAPs have the same cost: a case for optimization-aware qubit routing," in *2022 IEEE International Symposium on High-Performance Computer Architecture*, Apr. 2022, pp. 709–725, iSSN: 2378-203X.

[21] T.-A. Wu *et al.*, "A robust quantum layout synthesis algorithm with a qubit mapping checker," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022.

[22] H. Fan *et al.*, "Optimizing quantum circuit placement via machine learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 19–24.

[23] C.-Y. Huang *et al.*, "Reinforcement learning and dear framework for solving the qubit mapping problem," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[24] S. Park *et al.*, "A fast and scalable qubit-mapping method for noisy intermediate-scale quantum computers," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, New York, NY, USA, 2022, p. 13–18.

[25] C.-Y. Huang and W.-K. Mak, "CTQr: Control and timing-aware qubit routing," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 140–145.

[26] ——, "Efficient qubit routing using a dynamically-extract-and-route framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[27] W.-H. Lin and J. Cong, "ML-QLS: Multilevel quantum layout synthesis," *arXiv preprint arXiv:2405.18371*, 2024.

[28] B. Tan and J. Cong, "Optimality study of existing quantum computing layout synthesis tools," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1363–1373, 2020.

[29] S. Li, X. Zhou, and Y. Feng, "On constructing benchmark quantum circuits with known near-optimal transformation cost," 2023. [Online]. Available: https://arxiv.org/abs/2301.08932

[30] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[31] H. Zou, M. Treinish, K. Hartman, A. Ivrii, and J. Lishman, "LightSABRE: A lightweight and enhanced SABRE algorithm," *arXiv preprint arXiv:2409.08368*, 2024.

[32] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," *arXiv preprint arXiv:1902.08091*, 2019.

[33] R. Wille and L. Burgholzer, "MQT QMAP: Efficient quantum circuit mapping," in *Proceedings of the 2023 International Symposium on Physical Design*, ser. ISPD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 198–204. [Online]. Available: https://doi.org/10.1145/3569052.3578928

Monolithic 3D FPGA Design and Synthesis with Back-End-of-Line Configuration Memories

Faaiq Waqar[1*], Jiahao Zhang[2], Anni Lu[1], Zifan He[2], Jason Cong[2], Shimeng Yu[1*]
[1]Georgia Institute of Technology, Atlanta, GA; [2]University of California – Los Angeles, Los Angeles, CA
[*]Email: faaiq.waqar@gatech.edu, shimeng.yu@ece.gatech.edu

***Abstract*— This work presents a novel monolithic 3D (M3D) FPGA architecture that leverages stackable back-end-of-line (BEOL) transistors to implement configuration memory and pass gates, significantly improving area, latency, and power efficiency. By integrating n-type (W-doped In_2O_3) and p-type (SnO) amorphous oxide semiconductor (AOS) transistors in the BEOL, Si SRAM configuration bits are substituted with a less leaky equivalent that can be programmed at logic-compatible voltages. BEOL-compatible AOS transistors are currently under extensive research and development in the device community, with investment by leading foundries, from which reported data is used to develop robust physics-based models in TCAD that enable circuit design. The use of AOS pass gates reduces the overhead of reconfigurable circuits by mapping FPGA switch block (SB) and connection block (CB) matrices above configurable logic blocks (CLBs), thereby increasing the proximity of logic elements and reducing latency. By interfacing with the latest Verilog-to-Routing (VTR) suite, an AOS-based M3D FPGA implemented in 7 nm technology is demonstrated with 3.4× lower area–time squared product (AT^2), 27% lower critical path latency, and 26% lower reconfigurable routing block power on benchmarks including hyperdimensional computing and large language models (LLMs).**

Keywords*— *FPGA, Monolithic 3D (M3D), Amorphous Oxide Semiconductors, Reconfigurable Computing

I. Introduction

Over the past three decades, field-programmable gate arrays (FPGAs) have become increasingly popular in domains such as telecommunications (packet processing and network function virtualization) [1], system-on-chip development (prototyping and verification) [2], and hardware acceleration for AI/ML workloads at the edge and in the data center [3]. The advantage of FPGAs lies in their off-the-shelf reconfigurability, enabling the implementation of custom circuit designs using hardware description languages (HDLs) or high-level synthesis (HLS), thereby circumventing the substantial non-recurring engineering (NRE) costs—such as physical design, layout, and fabrication—associated with application-specific integrated circuit (ASIC) development. This adaptability extends their operational longevity in rapidly evolving markets where fixed-function ASICs quickly become obsolete. FPGAs can execute custom applications at over 10× lower power and with >3× runtime reduction compared to CPU implementations [4]. However, although modern FPGAs incorporate hardened macros such as RAMs, processor subsystems, and digital signal processing (DSP) units, designs implemented on an FPGA are still 9× larger and 3-6× slower than an equivalent built on an ASIC [5].

The principal cause of the power, performance, and area (PPA) disparity between FPGAs and ASICs is a byproduct of their key advantage: reconfigurability. Configuration memories implemented using SRAM enable/disable logical and signal propagation functions to emulate the data path in an ASIC. Modern FPGAs require a substantial number (~2,000-5,000) of configuration bits per tile (Section 2A). The low density and high static power consumption of SRAM cells significantly limit the density of FPGA designs; configuration memories can occupy >50% of a tile's area and account for ~12% of the total static power [6]. Additionally, the routing fabric comprises extensive networks of crossbars, multiplexers, buffers, and wires that dominate the dynamic (~75%) and static (~78%) power consumption [7]. We observe that reductions to the routing and reconfiguration overhead have significant implications on the PPA of FPGAs.

Monolithic 3D (M3D) integrated circuits, enabled by innovations in low-temperature materials processing, permit the use of multiple active tiers on a single substrate by building transistors in the back-end-of-line (BEOL). Among the most promising emerging transistor candidates are amorphous oxide semiconductor (AOS) transistors, owing in part to their commercial adoption in transparent thin-film channels for active display technology [8]. Beyond their BEOL compatibility and stackability, AOS transistors have ultra-low leakage ($<10^{-15}$ μA/μm) [9], high I_{on}/I_{off} ratio ($>4\times10^9$), and moderate electron mobility (~15-20 $cm^2/V\cdot s$) [10]. AOS transistors have found themselves at the forefront of research on charge-based memories [11], hybrid M3D standard cells [12], and BEOL power delivery [13]. Although vast literature exists on the applications of AOS transistors, their candidacy as a fully BEOL-compatible SRAM substitute has been largely unexplored due to the lack of a strong p-type AOS substitute with comparable hole mobility, which leads to a slower dynamic access speed critical for application in SRAM-based registers and cache memories. However, in FPGAs, SRAM configuration bits are stationary at runtime. Thus, the preferable criterion in a configuration SRAM bit cell is governed by its device stability, static power, and footprint, on which front BEOL-compatible AOS SRAMs can improve upon the Si counterpart [14].

This paper presents a design-space analysis of a novel M3D FPGA architecture with AOS transistor-based configuration SRAMs and multiplexed routing structures. Our proposed architecture bypasses the need for high-voltage conversion and delivery for programming (a density/reliability bottleneck in prior work on FPGAs employing emerging devices). To enable the precise quantitative study of an M3D FPGA with AOS device integration and its advantages over CMOS FPGAs in 7 nm technology, we develop an evaluation flow based on robust compact transistor models, a custom M3D-compatible version of COFFE [15] and Verilog-to-Routing (VTR) [16] to appraise PPA improvements on benchmarks targeting widely used kernels across deep learning, hyperdimensional computing (HDC), signal/data processing and LLM acceleration.

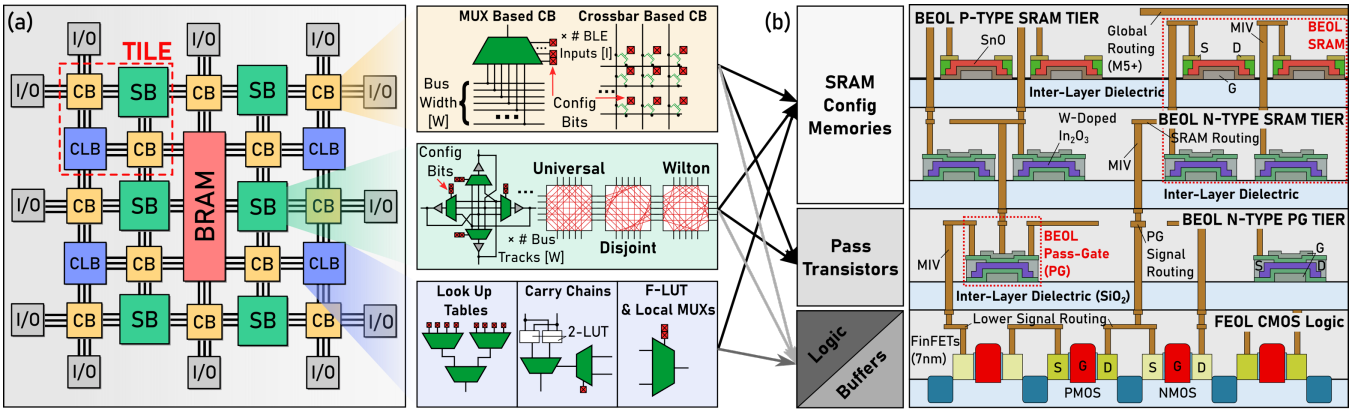


Fig. 1. (a) Island-based FPGA organization with breakdown of reconfigurable logic (CLB) and routing (Switch (SB), Connection (CB)) blocks. Extensive use of configuration bits and pass-transistors shown in red and green (b) proposed M3D FPGA organization of FEOL/BEOL devices and routing structures.

II. Background and Related Work

A. Conventional FPGA

Fig. 1a illustrates a conventional island-based FPGA design, composed of an array of tiles—modular units containing a Configurable Logic Block (CLB) and routing resources in the form of a Switch Block (SB) and pairs of Connection Blocks (CBs). Each CLB houses a cluster of N Basic Logic Elements (BLEs), as shown in Fig. 2a. The BLE datapath comprises a look-up table (LUT) that maps a K-input logical function, where inputs are fed to the selection pins of the LUT multiplexer. LUTs are interfaced with a CLB input mux grid to select operation inputs, flip-flops (FFs) to hold the logical expression's output, and, at times, Fracturable LUT (F-LUT) multiplexers to break down LUTs for higher utilization [17]. At the output, feedback multiplexers feed data recursively back to the input while carry chains sometimes expedite arithmetic within BLEs [18]. A global bus consisting of horizontal and vertical routing channels with multiple tracks (W) that span several tiles (L) connects to CLB input pins (I) through CBs, using either crossbar or mux-based interconnects. SBs are positioned at the intersections of routing channels to direct signals across various paths using routing switches, the number and placement of which depend on the topology (Fig. 1a) and the block's connection flexibility (F_s). All three principal blocks within a tile extensively utilize configuration bits fed into selection pins (and, thus, transistor gates) for routing and I/O selection and as inputs for LUT multiplexers. These interconnected blocks form a datapath, depicted in Fig. 2b, the longest of which determines the critical path delay (CPD) within a layout after place and route (P&R).

B. Amorphous Oxide Semiconductor Transistors

Amorphous oxide semiconductors (AOS) such as the indium oxide (In_2O_3) family are a class of materials that exhibit moderate electron mobility (~10–20 $cm^2/V{\cdot}s$) and can be fabricated at BEOL compatible temperatures (<400 ºC). In n-type AOS materials, conduction is primarily governed by donor defects such as oxygen vacancies. To suppress the formation of oxygen vacancies and enhance the stability of the film, dopants with high oxygen affinity, such as gallium (Ga), zinc (Zn), tin (Sn), or tungsten (W), are incorporated. Leading foundries and research institutes such as TSMC, Samsung, and IMEC are actively researching the enhancement of AOS transistors [14],[19]-[20]. Among various n-type AOS devices, W-doped In_2O_3 (IWO) was selected for this study because of its balanced characteristics of high on-state current density and threshold voltage stability [21]. Achieving p-type conduction in AOS is more challenging due to the localized nature of the valence band, involving acceptor defects like cation vacancies that facilitate hole conduction. Recently, TSMC reported a p-type tin-oxide (SnO) transistor with hole mobility as high as ~2 $cm^2/V{\cdot}s$ and a record I_{on} of 10~20 µA/µm [14]. Proposed AOS devices for M3D integration candidacy have been demonstrated down to channel lengths (L_{ch}) of 10 nm in n-type Ga/Zn-doped In_2O_3 (IGZO) [20], 20 nm n-type IWO [22], and 25 nm in p-type SnO [14]. The low-temperature fabrication process permits AOS transistors to be built above the Si logic layer without damaging the underlying devices, allowing active logic tiers to be stacked among the BEOL interconnect levels (Fig. 1b).

C. FPGAs using Emerging Memories and Devices

Prior works targeting emerging device integration for FPGA optimization fall into one of three categories: (1) the substitution of configuration memories and/or pass gates, (2) the truncation of multiplexing circuits, and (3) the expansion of block RAM (BRAM). This study belongs to (1), in which extensive work has been conducted on the use of resistive switching mechanisms in

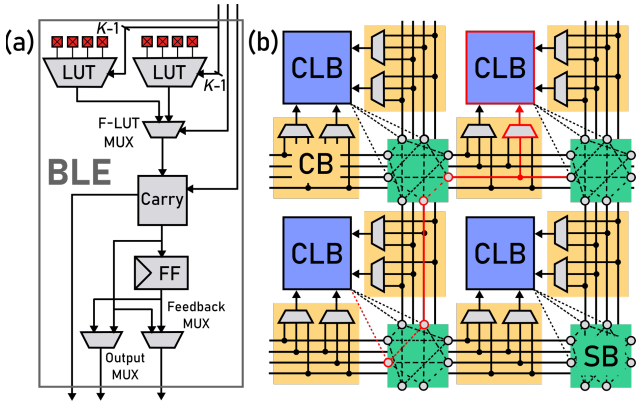


Fig. 2. (a) Composition of a basic logic element (BLE) in a CLB (b) Example of a critical path (red) in an island-based FPGA design, CLBs use LUTs to emulate logic and utilize SB and CB blocks to connect to one another. The placement of logical elements and routing paths is determined through a simulated annealing algorithm during place and route (P&R)

RRAM [23]-[25], FeFET [26], NEM relay [7],[27]-[28], and PCM [29]. Among RRAM-based FPGA studies, [23] proposes the replacement of pass-gates with RRAM in SBs/CBs and the use of 2-RRAM divider configuration memories to improve footprint 2-3× and reduce power by 20%, FPGA-RPI [24] develops routing/programming methodology and a BEOL place and route EDA flow with an adaptive buffering scheme for RRAM SBs/CBs that can reduce the programmable mesh area by 96% and improves performance by 55%, and [25] tapes-out a 1T-2R divider-based BEOL configuration memory over 289 tiles in 180 nm node reducing area by 57%. In [26], FeFET-based routing elements and current-based LUTs are proposed, reducing FPGA footprint by 8% and improving power-delay-product (PDP) by 5.8-16×, although the current-based LUT readout non-trivially increases static power consumption. Notable studies on NEM relay-based FPGAs include [27], which proposes relay routing structures and divider-based configuration memories to cut the area by 43.6% and latency by 28%, [28] that extends this further with face-to-face heterogeneous 3D integration to reduce latency by 42% over 2D CMOS, and [7] which demonstrates the functional operation of a 2-by-2 NEM relay for BEOL CBs. However, a challenge of employing the proposed emerging non-volatile memories (eNVMs) is the reliance on high-voltage programming, which necessitates the use of external circuitry and programming grid with high-voltage tolerance (such as thick-oxide I/O transistors) degrading density and reliability. Additionally, surface forces dominate over inertial forces in scaled NEM relays, hindering programming reliability. Strategic tradeoffs between prospective devices are further investigated in Section 3B.

III. BEOL Compatible Configuration Bits

A. Memory Cell Topology and Design

As noted in Section 1, configuration SRAM cells, distributed widely throughout CLBs, SBs, and CBs, can be quantitatively assessed by their ability to drive sufficient voltage and by their static power. For example, Xilinx transitioned to using larger middle-thickness oxide transistors for SRAM to lower the static power consumption in 40-45 nm technology nodes [30]. A cross-sectional view of our M3D placement of AOS SRAM cells is depicted in the upper tiers of Fig. 1b, above the lower metallization tiers (M1-M4). To minimize the footprint of each cell, two tiers are used to construct the AOS SRAMs, the upper for PMOS and the lower for NMOS. The storage node (Q) of the SRAM cell must connect to the local interconnect below to be accessed by front-end-of-line (FEOL) logic, which requires the signal to propagate through to M1 or M2. This transmission is done using monolithic interlayer vias (MIVs).

To evaluate the proposed AOS SRAM design, short channel (L_{ch}=25 nm) machine learning-assisted compact models [31] are utilized for calibrating the TCAD or experimental data reported from double-gated (DG) IWO (n-type) [22] and back-gated (BG) SnO (p-type) devices [14], which is then validated in SPICE simulation. The BG SnO FET compact model is calibrated directly from experimental data of a 25 nm channel length device. To determine the behavior of channel length scaling in DG IWO FETs with dual-gate operation, we model the structural changes made to an experimentally calibrated DG IWO FET in Sentaurus TCAD at 50 nm channel length, which captures the short channel effects of the AOS transistor. Then, an ML-assisted compact model is developed from Id-Vg, Id-Vd, and C-V data extracted from TCAD. Tile design parameters used to benchmark FPGA performance with/without AOS integration are set to $\{K, N, I, W, L, F_s\} = \{6, 10, 40, 150, 8, 3\}$.

B. Power, Stability, and Programmability

The bit cell layout of the proposed AOS SRAM cell is depicted in Fig. 3b. To minimize the AOS SRAM footprint, a bit cell is constructed in 2-tiers, where the inverter and feedback connections are made through pairs of MIVs, and separate metallization tiers are used for the WL and rail voltage connections. Fig. 3c depicts findings for the differences in static power between AOS and 7 nm Si-based (minimum-sized ASAP7 FinFET) configuration SRAMs. A 2:1:1 PU:PD:PG structure with a minimum width of 50 nm is used for the AOS SRAM design. When used at the same overdrive voltage (V_{SRAM}) of 0.8 V, AOS SRAMs can reduce static power consumption by 60.1% per bit cell over their CMOS counterpart, lowering the cumulative static power of configuration memories to ~5.3% of all static power on-chip. We compare the programming speed and voltage of both SRAMs and

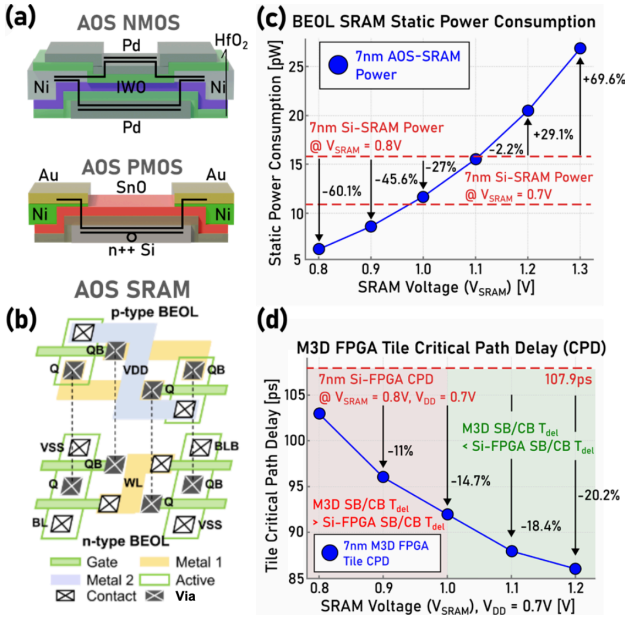


Fig. 3. (a) Device structure of double-gated (DG) n-type IWO and back-gated (BG) p-type SnO devices (b) 2-tier layout of BEOL compatible AOS SRAM (c) Comparison of AOS and CMOS SRAM static power as a function of V_{SRAM} (d) Comparison of conventional and M3D FPGA tile CPD as a function of V_{SRAM}

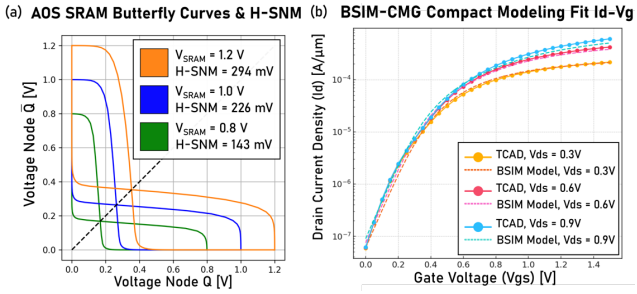


Fig. 4. (a) Butterfly curve and corresponding H-SNM of AOS SRAM vs. V_{SRAM} (b) BSIM-CMG model fitting for AOS NMOS-based pass-gate

TABLE I
Pass-Device Comparison Proposed for FPGA Reconfiguration

	RRAM [23]-[25]	NEM [27]-[28]	Si FeFET [26]	AOS FET 7nm	Si FET 7nm
R_{on}/R_{off}	10^3-10^6	> 10^9	10^6-10^7	10^9 (PG)	10^7 (PG)
V_{PRGM}	3.3 V	0.8 V*	5 V	0.7 V (SRAM)	0.7 V (SRAM)
T_{PRGM}	5 ns	< 10 ns	2 ns	440 ps	17 ps
Area	Small (BEOL)	Large (BEOL)	Small (FEOL)	Large (BEOL)	Large (FEOL)
Pass-Disturb	Moderate	High	Moderate	Low	Low

* Projection when scaled to 22 nm technology node metallization pitch

configuration memories proposed for SRAM substitution in prior works (Table I). Although the programming time of AOS SRAM bits is slower than that of conventional Si SRAM by a factor of 26×, the cell can still be sufficiently written using WL and BL pulses that do not need to exceed V_{DD} at 7 nm (0.7 V) thanks to the restorative feedback in SRAMs cross-coupled inverter structure, even when $V_{SRAM} > V_{DD}$. This eliminates the need for a programming grid for high-voltage applications, improving density and reliability while maintaining a programming speed 4-10× shorter than other eNVM technologies proposed for FPGA reconfigurable structures. AOS SRAMs demonstrate a sense noise margin (H-SNM) of 226 mV at 1.0 V, compared to 339 mV at 0.8 V in CMOS (Fig. 4a). The leftward shift of the metastable point and, thus, lower H-SNM arises due to a much weaker AOS PMOS with negative V_{TH}; however, an H-SNM >200 mV is sufficient tolerate noise injected at the storage node, such as that due to thermal noise and capacitive coupling at the pass-gate.

IV. BEOL Compatible Pass Gates for Routing

A. Equivalent Circuit Model and Topology

Fig. 5a shows that configuration memory substitution alone reduces the tile footprint by ~49%. However, there is considerable room for improvement as SBs and CBs still consume 17.6% and 13.8% of the footprint, respectively. Owing to the observation that the composure of routing blocks is almost entirely made from mux and/or crossbar structures consisting of either pass or transmission gates, the insertion of a tier of n-type IWO pass gate-based SB and CB above CLBs (Fig. 1b) could significantly reduce the cumulative footprint, which we refer to as a true M3D FPGA. The composition of this proposed M3D FPGA (with both BEOL configuration memories and pass gates) is also shown in Fig. 5a, demonstrating ~59% area reduction over the conventional CMOS tile. Furthermore, we sweep tile parameters that strongly influence tile area (Fig. 5b), demonstrating that the multi-tier M3D design improves the footprint scalability of logic complexity in CLBs. However, in tiles with large routing bus widths, congestion in the BEOL can limit footprint reduction.

B. Timing, Area, and Power

To measure the performance and power reductions of AOS routing pass gate (PG) structure, we modify the DG IWO FET TCAD structure to capture the device physics under 7 nm design rule (L_{ch}=18 nm) and model it with BSIM-CMG for SPICE simulation (Fig. 4b). Base logic input signals and buffers are connected to AOS pass gates through MIVs, for which we use

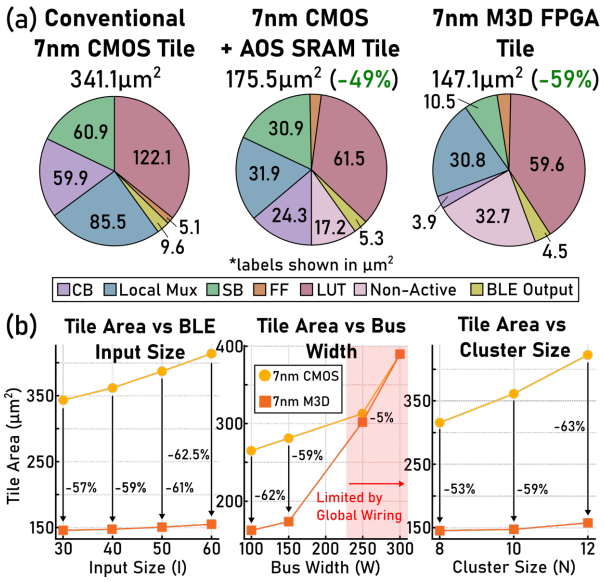


Fig. 5. (a) Composition of 7 nm CMOS, CMOS +AOS SRAM, and M3D FPGA designs (b) Parametric area comparison of CMOS and M3D tiles

parasitics of 0.18 fF and 96 Ω per MIV [32]. We assume the implementation of BEOL pass gates above the signal routing layers for CLB logic and buffers (M1-M2) and dedicate a separate signal routing tier for PGs (M3) using ASAP7 M1-M3 parasitics (131.2 Ω/µm, 0.23 fF/µm).

In Fig. 3d, we compare the tile level critical path delay (CPD) in CMOS and M3D FPGAs as a function of the SRAM voltage (V_{SRAM}) under specified design parameters modeled based on the Xilinx-7 architecture. Cumulative critical path delay is reduced at the same V_{SRAM} (0.8 V) by ~8% thanks to shorter routing/load RC. However, this reduction is limited by the lower drive current and higher source/drain capacitance found in AOS transistors due to the large overlap regions and lower channel electron mobility [11]. We find that overdriving the AOS pass gates to/beyond 1.0 V also improves SB/CB performance while reducing the configuration memory static power by 27%. With this in mind, we consider the split of the rail voltages of SRAM configuration bits between those set for CLBs and those used to drive AOS pass gates in SBs and CBs (V_{SRAM}, $V_{SRAM\text{-}SCB}$).

Buffers and the reconfigurable mesh consume the most power in FPGAs: ~82% of static and ~68% of dynamic system power [7]. We observe that a disadvantage of using pass-gates over transmission gates in CMOS is their inability to transmit a signal in full swing, limited by the threshold voltage of the pass device. This is supplemented by a level-restorer, which pulls up the voltage to digital-circuit readable levels. However, restorers increase power and latency. Given the larger bandgap in AOS materials, it is possible to exceed V_{SRAM} voltages used on CMOS pass gates without severely degrading the reliability. A $V_{SRAM\text{-}SCB} \geq V_{DD} + V_{TH}$, such as 1.2 V, removes the need for a level-restorer while increasing SRAM static power consumption by only 29.1% in AOS SB and CB configuration bits over CMOS. However, this increase is exceeded by the cumulative power reduction in the M3D design. To evaluate this modification, we test SB and CB structures made of AOS and CMOS devices in SPICE (Fig. 6a). The average power at 250MHz is given by:

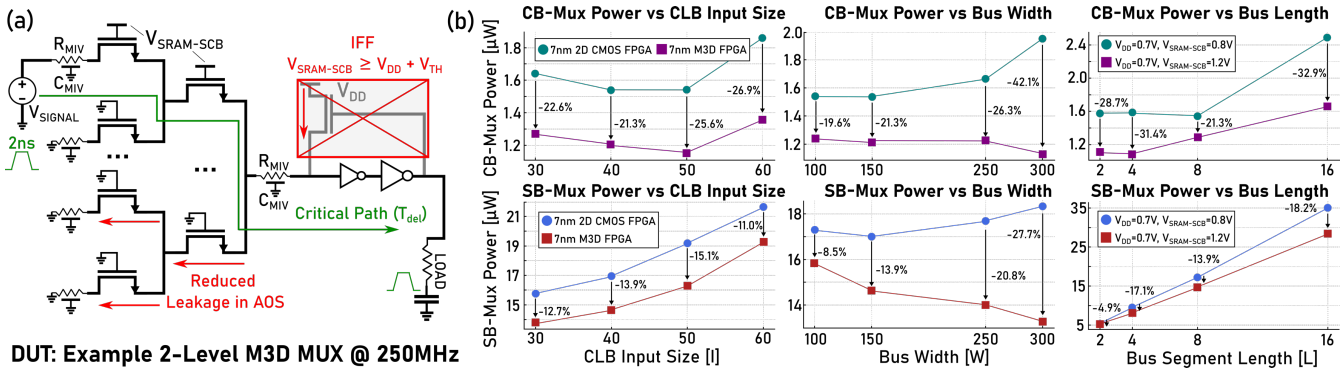


Fig. 6. (a) Design under test (DUT) for an example 4:1 mux with overdriven IWO pass gates, enabling removal of level-restorers (b) parametric sweep of bus parameters vs. SB/CB mux power consumption. Level-restorer removal and low leakage in AOS devices improve the efficiency of reconfigurable routing.

$$P_{avg} = \int_0^{4ns} V_{dd} \times I_{src} dt \Big/ 4ns + N_{sram} \times P_{sram} \qquad (1)$$

Where I_{src} is the current supplied by the voltage source, N_{sram} is the number of configuration bits in the DUT, and P_{sram} is the static power of the AOS SRAM cell at $V_{SRAM-CSB}$. Parameters defining the interfacial widths between bus connections and bus dimensionality are swept as CB and SB blocks steer connectivity through/off the routing bus interface (Fig. 6b). For reference, a single SB-Mux or CB-Mux refers to a modular building block of an SB or CB, in which a tile contains multiplicities of W (× SB-Mux) and I (× CB-Mux) respectively. We observe that, on average, switching to AOS pass gates and overdriving $V_{SRAM-SCB}$ to 1.2 V reduces the power consumption of SBs by ~13.7% and CBs by ~26% over CMOS at V_{SRAM} of 0.8 V, closing the gap between ASIC and FPGA power efficiency.

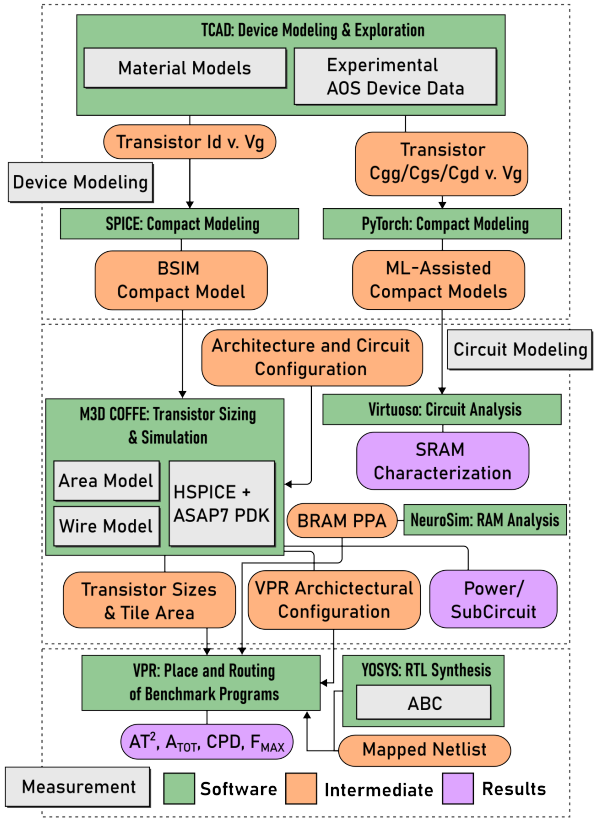


Fig. 7. Benchmarking methodology of 7 nm CMOS and M3D FPGAs

V. Evaluation Methodology

A. M3D Architectural Simulation of Xilinx-7 FPGA

Fig. 7 visualizes this work's evaluation flow of devices, circuits, and architectural measurement. To deal with the novel M3D placement structure and optimization of AOS PG widths and to develop architectural level parameters of our design, we develop a custom version of the tool COFFE [15] that can interface with the latest version of the academic Verilog-to-Routing (VTR) suite that is widely used for FPGA architectural benchmarking [16]. Circuit simulations are carried out in Synopsys HSPICE and Cadence Virtuoso using the compact device models described in prior sections and the ASAP7 PDK. Cost optimization is exhaustively performed for benchmarked circuits and tiles using the area-delay product. The Xilinx-7 architectural configuration file from VTR 8 is modified using M3D COFFE to reflect the interconnect, area, and delay changes in 7 nm technology and using the substitution of AOS transistors. A custom version of NeuroSim [33] is employed to estimate PPA parameters in a 32kB 7 nm FinFET dual-bank block RAM. VPR is used for CPD, total logic area utilization (A_{TOT}), maximum operation frequency (F_{MAX}), and AT^2.

B. Benchmarking on Diverse FPGA Workloads

The growing demand for efficient processing of diverse workloads, including widely used kernels in deep learning, large language models (LLMs), hyperdimensional computing (HDC), and signal and data processing (Table II), drives the exploration of FPGA accelerators to exploit their inherent parallelism and energy efficiency. To evaluate performance across different computational patterns, we select FFT [34], CNN [35], and GEMM [36] as representative compute-intensive kernels commonly used in digital signal processing and machine learning. We include sparse matrix-vector multiplication (SPMV) for memory-bound workloads, which benefit from the custom memory hierarchies and parallelism available on FPGAs, often outperforming GPUs and general-purpose processors in energy efficiency. Additionally, AES [37] and Huffman Coding [38] represent workloads with complex control logic and multiple sub-kernels, typical of data encryption and compression tasks. Together, these benchmarks assess the FPGA's ability to handle a wide range of compute-intensive, memory-bound, and control-intensive applications.

TABLE II
Key Kernel Benchmarking Parameters

Benchmark	Parameters	Input Type
FFT	Input Data Size n = 64	32b FP
Convolution	Input Size = 128×228×228 Kernel Size = 128×128×5×5 Output Size = 128×224×224 5×5 Convolution Unit Number = 16	8b INT
GEMM	Matrix Size = 512×512 Systolic Array Size = 8×8	8b INT
SPMV	Single HiSparse [39] SPMV Cluster	32b FP
AES Enc. & Dec.	Key Size = 256b	128b INT
Huffman Encoder	Input Symbol Size = 256 Max Tree-Construction Depth = 64 Max Rebalanced Tree Depth = 27	10b UINT 32b UINT

VI. Benchmarking Results

Comparisons of the system level CPD, A_{TOT}, and AT^2 and the geometric mean for benchmarks in Table II are shown in Fig. 8, derived from VPR. We assess two systems in our study, both based on the Xilinx-7 architecture. The first is a baseline conventional CMOS design calibrated using 7 nm FinFET models from ASAP7 using COFFE with rails V_{DD}=0.7 V, V_{SRAM}=0.8 V. The second is our M3D FPGA design with AOS SRAM and PG substitution, which is calibrated in M3D COFFE utilizing our AOS BSIM-CMG model and ASAP7 with rails V_{DD}=0.7 V, V_{SRAM}=0.8 V and $V_{SRAM\text{-}SCB}$=1.2 V. Benchmarking findings indicate that our M3D FPGA design can decrease A_{TOT} by >52%, CPD by 11.9% - 30%, and AT^2 by 63.6% - 77.1%. From prior works, we estimate that this reduction in performance and area would close the gap between ASICs and FPGAs down to 2.05-4.37× and 4.2× based on findings in [5].

Additionally, in Fig. 9a, we observe changes to the average distribution of routing utilization across relevant benchmarks at different localized and unlocalized segment lengths. In the M3D design, we find that the optimized layout post-P&R has a higher favorability for longer routing segments due to the reduced cost of global routes. By translation, as is seen in Fig. 9b, which visualizes the congestion and critical path map from the AES benchmark, the congestion of nets becomes far more distributed

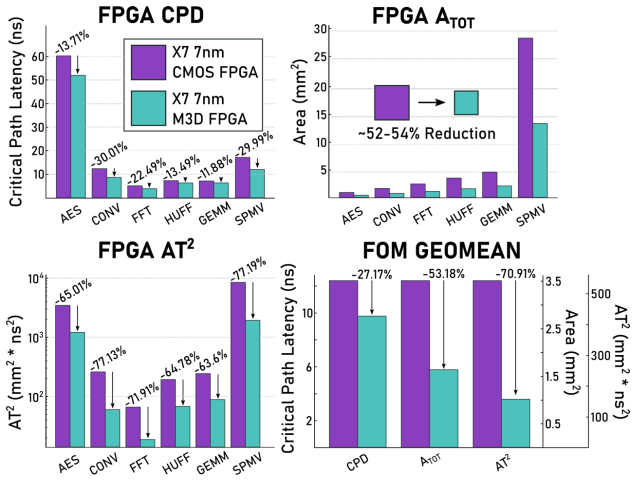


Fig. 8. Comparison of critical path delay, cumulative logic area, AT^2, and geometric mean across hyperdimensional computing benchmarks

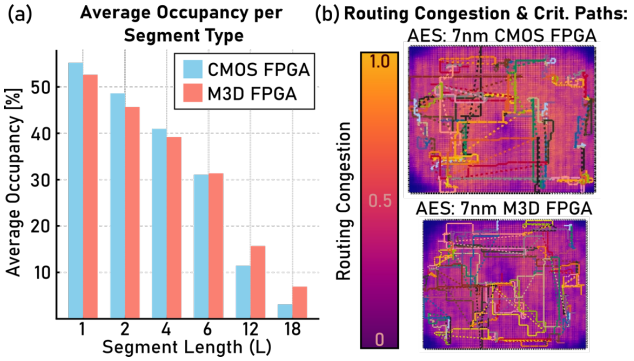


Fig. 9. (a) Comparison of routing occupancy, (b) congestion and critical paths across benchmarks in Xilinx-7 based CMOS and M3D FPGAs. M3D has a higher affinity for longer segments, reducing the signal congestion

on-chip. This can help delocalize hotspots and improve the placement proximity of hard macros in FPGAs.

Given the rapid growth of LLM parameter sizes, we target a GPT-2 medium model accelerator implemented on the AMD Xilinx Versal VPK180 FPGA and estimate the utilization area based on the resource allocation relative to the total chip area of the VPK180. Delay is calculated based on the latency of critical paths after placement and routing using Vivado 2023.2. Aiming to transfer the same design to an M3D FPGA, we estimate the area and delay, assuming identical resource utilization. We observe a 63.7% improvement in AT^2 when translating the design from the VPK180 to an M3D equivalent, achieving significant reductions in area and delay (Table III).

TABLE III
Comparison of LLM (GPT-2) Implementation on M3D FPGA

System	Versal VPK 180	7 nm M3D FPGA
Area	736.0 mm²	423.8 mm² (-42.4%)
Delay	4.17 ns	3.31 ns (-20.6%)
AT^2	12798.9 mm²×ns²	4643.2 mm²×ns² (-63.7%)

VII. Conclusions

This work presents a design-space analysis of a novel M3D FPGA architecture with AOS device-based configuration SRAMs and routing pass gate structures used in switch and connection blocks. A CAD flow for exploring the proposed M3D design that can optimize the placement and sizing of AOS devices is developed to evaluate performance on a suite of compute-intensive hyperdimensional computing tasks ranging from 170×170 to 372×372 logical blocks per net. The proposed M3D FPGA demonstrates improvements of up to 30% lower delay, 54% lower system area, and 77% lower AT^2. By transitioning to AOS devices with reduced static power, monolithically stacking devices to create multiple active tiers, and decreasing the overhead of interconnects, the proposed design cuts the PPA disparity between ASICs and FPGAs, enhancing the applicability of FPGA acceleration across a diverse set of workloads.

VIII. Acknowledgments

This work is supported by PRISM, an SRC/DARPA JUMP 2.0 center. The authors thank S. Jung from Georgia Tech and S. More from Univ. of Toronto for guiding discussions on device compact modeling and open-source FPGA EDA tools.

REFERENCES

[1] P. Sikka, A. R. Asati, and C. Shekhar, "Speed optimal FPGA implementation of encryption algorithms," *Microprocess. Microsyst.*, vol. 79, 2020.

[2] Y. Huang, Y. F. Yin, C. J. Hsu, T. B. Huang, and T. M. Chang, "SoC HW/SW verification and validation," in *Proc. 16th Asia and South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2011, pp. 297-300.

[3] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Comput. Appl.*, vol. 32, no. 4, pp. 1109-1139, 2020.

[4] A. Boutros and V. Betz, "FPGA architecture: Principles and progression," *IEEE Circuits Syst. Mag.*, vol. 21, no. 2, pp. 4-29, 2021.

[5] A. Boutros, S. Yazdanshenas, and V. Betz, "FPGA vs. ASIC efficiency gaps for CNN inference," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1-23, 2018.

[6] I. Herrera-Alzu and M. Lopez-Vallejo, "Design techniques for Xilinx Virtex FPGA configuration memory scrubbers," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 1, pp. 376-385, 2013.

[7] C. Chen, W. S. Lee, R. Parsa, S. Chong, J. Provine, J. Watt, R. T. Howe, H. S. P. Wong, and S. Mitra, "Nano-electro-mechanical relays for FPGA routing: Experimental demonstration and a design technique," in *Proc. Design, Autom. Test Europe Conf. & Exhib. (DATE)*, Mar. 2012, pp. 1361-1366.

[8] A. Nathan, S. Lee, S. Jeon, and J. Robertson, "Amorphous oxide semiconductor TFTs for displays and imaging," *J. Display Technol.*, vol. 10, no. 11, pp. 917-927, 2014.

[9] S. Datta, E. Sarkar, K. Aabrar, S. Deng, J. Shin, A. Raychowdhury, S. Yu, and A. Khan, "Amorphous Oxide Semiconductors for Monolithic 3D Integrated Circuits," in *Proc. 2024 IEEE Symp. VLSI Technol. Circuits*, Jun. 2024, pp. 1-2.

[10] T. Kamiya and H. Hosono, "Material characteristics and applications of transparent amorphous oxide semiconductors," *NPG Asia Mater.*, vol. 2, no. 1, pp. 15-22, 2010.

[11] S. Liu, K. Jana, K. Toprasertpong, J. Chen, Z. Liang, Q. Jiang, S. Wahid, S. Qin, W. C. Chen, E. Pop, and H. S. P. Wong, "Design guidelines for oxide semiconductor gain cell memory on a logic platform," *IEEE Trans. Electron Devices*, 2024.

[12] J. Kwak, G. Choe, and S. Yu, "Design-technology co-optimization for stacked nanosheet oxide channel transistors in monolithic 3D IC design," *IEEE Trans. Nanotechnol.*, 2024.

[13] S. Deng, J. Kwak, J. Lee, D. Chakraborty, J. Shin, O. Phadke, S. G. Kirtania, C. Zhang, K. A. Aabrar, S. Yu, and S. Datta, "Monolithic switched-capacitor DC–DC converters using BEOL-compatible oxide power transistors and superlattice MIM capacitors," *IEEE Trans. Electron Devices*, 2024.

[14] C. C. Wang, C. C. Kuo, C. H. Wu, A. Lu, H. Y. Lee, C. F. Hsu, P. J. Tzeng, T. Y. Lee, F. R. Hou, M. H. Chang, and S. C. Lai, "P-type SnO semiconductor transistor and application," in *Proc. 2024 IEEE Symp. VLSI Technol. Circuits*, Jun. 2024, pp. 1-2.

[15] S. Yazdanshenas and V. Betz, "COFFE 2: Automatic modelling and optimization of complex and heterogeneous FPGA architectures," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 1-27, 2019.

[16] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J. P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. Walker, and H. Zeng, "VTR 8: High-performance CAD and customizable FPGA architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 2, pp. 1-55, 2020.

[17] J. Luu, J. Rose, and J. Anderson, "Towards interconnect-adaptive packing for FPGAs," in *Proc. 2014 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2014, pp. 21-30.

[18] S. Hauck, M. M. Hosler, and T. W. Fry, "High-performance carry chains for FPGAs," in *Proc. 1998 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Mar. 1998, pp. 223-233.

[19] D. Ha, Y. Lee, S. Yoo, W. Lee, M. H. Cho, K. Yoo, S. M. Lee, S. Lee, M. Terai, T. H. Lee, and J. H. Bae, "Exploring innovative IGZO-channel based DRAM cell architectures and key technologies for sub-10nm node," in *Proc. 2024 IEEE Int. Memory Workshop (IMW)*, May 2024, pp. 1-4.

[20] M. Si, Z. Lin, Z. Chen, X. Sun, H. Wang, and P. D. Ye, "Scaled indium oxide transistors fabricated using atomic layer deposition," *Nature Electron.*, vol. 5, no. 3, pp. 164-170, 2022.

[21] K. A. Aabrar, H. Park, S. G. Kirtania, E. Sarkar, M. A. Al Mamun, S. Deng, C. Zhang, G. B. Rayner, K. Cho, and S. Datta, "On the reliability of high-performance dual gate (DG) W-doped In2O3 FET," in *Proc. 2024 IEEE Symp. VLSI Technol. Circuits*, Jun. 2024, pp. 1-2.

[22] H. Ye, J. Gomez, W. Chakraborty, S. Spetalnick, S. Dutta, K. Ni, A. Raychowdhury, and S. Datta, "Double-gate W-doped amorphous indium oxide transistors for monolithic 3D capacitorless gain cell eDRAM," in *Proc. 2020 IEEE Int. Electron Devices Meeting (IEDM)*, Dec. 2020, pp. 28-3.

[23] S. Tanachutiwat, M. Liu, and W. Wang, "FPGA based on integration of CMOS and RRAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 11, pp. 2023-2032, 2010.

[24] J. Cong and B. Xiao, "FPGA-RPI: A novel FPGA architecture with RRAM-based programmable interconnects," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 4, pp. 864-877, 2013.

[25] Y. Y. Liauw, Z. Zhang, W. Kim, A. El Gamal, and S. S. Wong, "Nonvolatile 3D-FPGA with monolithically stacked RRAM-based configuration memory," in *Proc. 2012 IEEE Int. Solid-State Circuits Conf.*, Feb. 2012, pp. 406-408.

[26] X. Chen, K. Ni, M. T. Niemier, Y. Han, S. Datta, and X. S. Hu, "Power and area efficient FPGA building blocks based on ferroelectric FETs," *IEEE Trans. Circuits Syst. I*, vol. 66, no. 5, pp. 1780-1793, 2018.

[27] C. Chen, R. Parsa, N. Patil, S. Chong, K. Akarvardar, J. Provine, D. Lewis, J. Watt, R. T. Howe, H. S. P. Wong, and S. Mitra, "Efficient FPGAs using nanoelectromechanical relays," in *Proc. 18th ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2010, pp. 273-282.

[28] C. Dong, C. Chen, S. Mitra, and D. Chen, "Architecture and performance evaluation of 3D CMOS-NEM FPGA," in *Proc. Int. Workshop Syst. Level Interconnect Prediction*, Jun. 2011, pp. 1-8.

[29] Y. Chen, J. Zhao, and Y. Xie, "3D-NonFAR: Three-dimensional non-volatile FPGA architecture using phase change memory," in *Proc. 16th ACM/IEEE Int. Symp. Low Power Electron. Design*, Aug. 2010, pp. 55-60.

[30] M. Klein, "Power consumption at 40 and 45 nm," White Paper, no. 298, pp. 1-21, 200.

[31] C. T. Tung, S. Salahuddin, and C. Hu, "A SPICE-compatible neural network compact model for efficient IC simulations," in *Proc. 2024 Int. Conf. Simulation Semiconductor Processes Devices (SISPAD)*, Sep. 2024, pp. 01-04.

[32] Y. Luo, S. Dutta, A. Kaul, S. K. Lim, M. Bakir, S. Datta, and S. Yu, "A compute-in-memory hardware accelerator design with back-end-of-line (BEOL) transistor-based reconfigurable interconnect," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 12, no. 2, pp. 445-457, 2022.

[33] P. Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3067-3080, 2018.

[34] I. S. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of fast Fourier transforms for real-time signal and image processing," *IEE Proc. Vision Image Signal Process.*, vol. 152, no. 3, pp. 283-296, 2005.

[35] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating CNN inference on FPGAs: A survey," *arXiv preprint arXiv:1806.01683*, 2018.

[36] R. Xu, S. Ma, Y. Guo, and D. Li, "A survey of design and optimization for systolic array-based DNN accelerators," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 1-37, 2023.

[37] A. M. Abdullah, "Advanced encryption standard (AES) algorithm to encrypt and decrypt data," *Cryptogr. Netw. Secur.*, vol. 16, no. 1, p. 11, 2017.

[38] A. Moffat, "Huffman coding," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1-35, 2019.

[39] Y. Du, Y. Hu, Z. Zhou, and Z. Zhang, "High-performance sparse linear algebra on HBM-equipped FPGAs using HLS: A case study on SpMV," in *Proc. 2022 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2022, pp. 54-64.

Introducing Instruction-Accurate Simulators for Performance Estimation of Autotuning Workloads

Rebecca Pelke, Nils Bosbach, Lennart M. Reimann, Rainer Leupers
Institute for Communication Technologies and Embedded Systems
RWTH Aachen University, Germany
{pelke, bosbach, lennart.reimann, leupers}@ice.rwth-aachen.de

***Abstract*—Accelerating Machine Learning (ML) workloads requires efficient methods due to their large optimization space. Autotuning has emerged as an effective approach for systematically evaluating variations of implementations. Traditionally, autotuning requires the workloads to be executed on the target hardware (HW). We present an interface that allows executing autotuning workloads on simulators. This approach offers high scalability when the availability of the target HW is limited, as many simulations can be run in parallel on any accessible HW.**

Additionally, we evaluate the feasibility of using fast instruction-accurate simulators for autotuning. We train various predictors to forecast the performance of ML workload implementations on the target HW based on simulation statistics.

Our results demonstrate that the tuned predictors are highly effective. The best workload implementation in terms of actual run time on the target HW is always within the top 3 % of predictions for the tested x86, ARM, and RISC-V-based architectures. In the best case, this approach outperforms native execution on the target HW for embedded architectures when running as few as three samples on three simulators in parallel.

***Index Terms*—Autotuning, TVM, gem5, cache optimization**

I. Introduction

The optimization of Machine Learning (ML) models is crucial due to their high computational demands. Traditional analytical methods often fall short given the huge search space for optimal implementations on modern CPU architectures. To address this challenge, autotuning has emerged as a powerful approach. Autotuning systematically evaluates multiple implementations of the same workload using mathematical models or ML techniques to guide subsequent selections of implementations [1], [2]. Apache Tensor Virtual Machine (TVM) [3], a well-known ML compiler framework, implements several autotuning strategies [4], [5]. Autotuning typically requires execution on real hardware, which introduces non-determinism due to factors such as the system load [6], cache collisions [7], thermal throttling [8], and frequency and voltage scaling [9].

To mitigate these issues, each benchmark is executed multiple times, outliers are removed, cooldown periods are inserted, and caches are flushed before each repetition. Consequently, benchmarking a single implementation takes significantly longer than its actual run time. This is time-consuming, especially when only limited hardware (HW) devices are available. This paper presents the following two main contributions:

Contribution Ⓘ: Simulator Interface - We present an interface allowing autotuning workloads to be executed on simulators rather than real hardware. Figure 1 Ⓘ illustrates this approach. We extract the TVM tasks and provide them as executables for the target architecture through a simulator interface. The number of parallel tasks is configurable, each running in a separate instance of the simulator. Potential scenarios that profit from our **simulator interface** are:

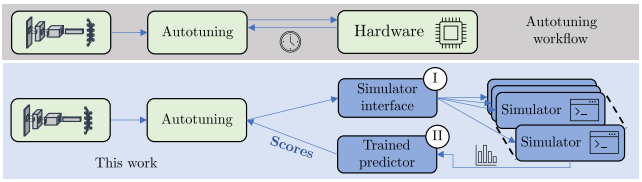


Fig. 1: The proposed simulator interface Ⓘ and the score predictor approach using instruction-accurate simulators Ⓘ

- The HW is not yet available, e.g., for pre-silicon software (SW) development.
- The embedded HW is only available in limited quantities, making parallel execution of simulations on high-performance computers faster than native execution.
- Other metrics besides run time should be optimized.

Contribution Ⓘ: Score Predictor - We aim to demonstrate that instruction-accurate simulators can be used for performance analysis, using autotuning workloads as an example. *To the best of our knowledge, we are the first to show how performance analysis can be conducted with fast instruction-accurate simulators.* Many open-source implementations of instruction-accurate simulators exist for different architectures, e.g., QEMU [10] or gem5 [11]. Figure 1 Ⓘ illustrates this idea. A predictor gets statistics from an instruction-accurate simulator as input and calculates a score based on reference values measured on real hardware. The score is then returned to the autotuning framework.

Since an instruction-accurate simulator does not provide accurate timing, we do **not** aim to predict execution latencies. Instead, our approach utilizes scores to evaluate and compare different implementations of the same workload. These scores are essential for guiding the autotuning process but are not suitable for comparing different types of workloads. We employ multiple predictors for this task: Multiple Linear Regression (MLR) [12], Deep Neural Networks (DNNs) [13], Bayesian optimization [14], and XGBoost [15]. We tune and compare them to identify the most accurate predictions for common ML kernels. We evaluate our approach on different CPU architectures, namely x86, ARM, and RISC-V.

II. Background and Related Work

A. Autotuning in TVM

Apache TVM [3] is an open-source ML compiler framework designed to optimize computations across various hardware platforms. In TVM, each operation, called *kernel* in this work, can be expressed in different abstractions, e.g., in Tensor Expression (TE) or in Tensor Intermediate Representation (TIR). For example, a kernel can be a Neural Network (NN) layer. TVM enables the application of *transformation primitives* to optimize kernels for diverse target architectures.

Similar to Halide [16], TVM distinguishes between the *compute* operation, which defines the functional behavior of the kernel, and the *schedule*, which defines its implementation. This distinction allows a single computation to have multiple schedules. The set of all possible schedules of an operation is called *design space*. Large design spaces and complex hardware behavior make analytical approaches impractical for finding optimal schedules for ML kernels. Autotuning addresses this issues by empirically evaluating multiple schedules directly on the target hardware. TVM offers three autotuning concepts: the AutoTVM framework, the Auto-Scheduler (also called Ansor [4]), and the Meta-Scheduler. This work focuses on AutoTVM and the Auto-Scheduler since they operate on the same input representation, namely TE.

Listing 1 provides an example of a TE compute operation definition. Tensor C (size $N \times M$) contains the result of a Matrix Matrix Multiplication (MMM) between matrices A (size $N \times L$) and B (size $L \times M$).

```
k = te.reduce_axis((0, L), name="k")
C = te.compute((N, M), lambda i, j: \
    te.sum(A[i,k]*B[k,j], axis=k), name="matmul")
```

Listing 1: Definition of a MMM compute operation in TE

AutoTVM requires users to define templates with tunable parameters, such as loop tiling factors. AutoTVM then navigates this search space by executing configurations on real hardware and measuring the run time. It requires some expertise to define a useful schedule template. However, predesigned templates for common operators can be found in the repository. As an example, Listing 2 illustrates how a *split* primitive of a single axis can be described using AutoTVM.

```
s = te.create_schedule(C.op)
y,_ = s[C].op.axis
# Define schedule template
cfg = autotvm.get_config()
cfg.define_split("split_y", y, num_outputs=2,...)
# Apply schedule
yo, yi = cfg["split_y"].apply(s, C, y)
```

Listing 2: Definition of a scheduling template for AutoTVM

In contrast to AutoTVM, the Auto-Scheduler automates the schedule generation process without requiring manual template definitions, making it more suitable for non-experts.

Figure 2 illustrates the workflows of both the Auto-Scheduler and AutoTVM. Unlike the manually defined search space in AutoTVM, the Auto-Scheduler uses *sketches* generated from the kernel's Directed Acyclic Graph (DAG) through

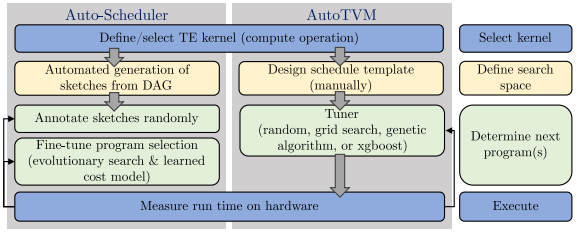


Fig. 2: Autotuning using Auto-Scheduler and AutoTVM

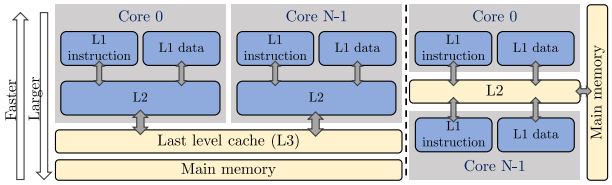


Fig. 3: Typical cache hierarchies of modern CPUs

predefined derivation rules. A sketch basically contains nested loops with placeholders, which are filled during an *annotation* phase. During annotation, loop axes can also be marked for parallelization, unrolling, or vectorization. Implementation details can be found in [4]. In contrast to the Auto-Scheduler, AutoTVM relies on tuners responsible for selecting subsequent programs based on selectable tuning algorithms.

B. Cache Hierarchy

ML operations require cache optimizations to achieve high performance [17], [18]. Modern CPUs have hierarchical memory systems with multiple cache levels (L1, L2, L3) as shown in Figure 3. The hierarchy can vary between different CPUs [19]. The L1 cache is usually divided into Data (L1D) and Instruction (L1I) cache. Higher-level caches are often shared among the cores. Most CPUs use N-way set-associative caches, where each memory address maps to one of N *ways* within a specific *set*. In Linux environments, information on the cache hierarchy can be accessed via the `sysfs`.

C. The gem5 Simulator

gem5 [11] is an open-source Full System Simulator (FSS) used in computer architecture research, supporting different architectures like x86, ARM, and RISC-V. It provides different abstraction levels. In gem5's *atomic* mode, memory accesses are executed within a single transaction. The requester of the memory access is blocked until the access is completed. In the *timing* mode, gem5 provides detailed simulation of memory access timing, including latency, queuing delays, and bandwidth constraints. gem5 also offers different CPU models. The `SimpleCPU` does not model a pipeline, which makes it fast but not very accurate in terms of timing. It can be used in both, atomic and timing mode. gem5 further provides an `InOrderCPU` and `O3CPU` (out-of-order CPU), which are both equipped with a CPU pipeline model. In addition, gem5 distinguishes between the *full-system* mode and the *system call emulation* mode. The system call emulation mode aims to simulate user-space programs. It intercepts system calls of the target software and handles them on the host. The

```python
class SimulatorRunner(Runner):
    def __init__(self, n_parallel=16, ...) :
        super(SimulatorRunner, self).__init__(...)
    def run(self, measure_inputs, build_results):
        # Build executables
        # Run executables in parallel on simulator
        return [AutoTVMRes0, AutoTVMRes1, ...]
```

Listing 3: Custom run function for AutoTVM flow

```python
@tvm._ffi.register_func(<func_name>, override=True)
def local_run(inputs, build_results, ...) :
    # Build executables
    # Run executables in parallel on simulator
    return [AutoSchedRes0, AutoSchedRes1, ...]
```

Listing 4: Custom run function for Auto-Scheduler flow

system call emulation mode is faster than the full-system mode and focuses on software testing, as it does not simulate the Operating System (OS) and thus cannot capture any OS-specific behavior.

III. Implementation

In the first part of this chapter, we explain the implementation of the simulator interface. In the second part, we use this interface to connect TVM's autotuning with an x86, an ARM, and a RISC-V-based instruction-accurate simulator.

A. Simulator Interface

The simulator interface allows for executing autotuning implementations on user-level or syscall-emulation simulators instead of real hardware. This enables parallel execution of different implementations. TVM's autotuning requires a *builder* and a *runner*. The builder generates an object file that contains the compiled functionality of the workload. The runner executes the workload using the `tvm::runtime`.

For a simulator, a standalone executable is needed. The executable prepares the input tensors, allocates space for the output tensors, and calls the compiled workload. To integrate this behavior into AutoTVM, we implement a custom runner called `SimulatorRunner` by inheriting from the `Runner` class (see Listing 3). When the `run` function of the custom runner is called, it generates a main function, compiles it, and links it against the compiled object file. A function called `simulator_run` is called with the path to the executable. This function serves as a simulator interface and can be overwritten to use a simulator for execution. The return value needs to be a score quantifying the performance of the workload, e.g., the run time. A parameter named `n_parallel` defines how many simulators can be instantiated in parallel.

To integrate a simulator into the Auto-Scheduler, we override a function in TVM's function registry called `auto_scheduler.local_runner.run`. This can be seen in Listing 4. The return value is a list of `AutoSchedResults` containing scores. The remaining implementation works in the same way as for AutoTVM.

When generating an object file for a ML kernel, the kernel is optimized and lowered through TVM. LLVM is used for code generation. To produce cross-compiled object files, it is possible to specify an LLVM triple in TVM.

```python
def conv2d(N,H,W,CO,CI,KH,KW,...)
    ifm=te.placeholder((N,CI,H,W),...)
    weights=te.placeholder((CO,CI,KH,KW),...)
    bias=te.placeholder((N,CO,1,1),...)
    conv=topi.nn.conv2d_nchw(ifm,weights,...)
    ofm=topi.nn.relu(conv+bias)
    # Return values: transferred as DLPack tensors
    return [ifm,kernel,bias,ofm]
```

Listing 5: Conv2D+Bias+ReLU kernel definition in TVM

Listing 5 shows the definition of a Conv2D+Bias+ReLU kernel as an example. The tensor shapes and parameters are passed as command line arguments to the executable.

B. gem5 Simulator Setup

To accurately predict timing, a cycle-accurate simulator is needed. However, cycle-accurate simulators often suffer from slow simulation speed. Additionally, obtaining cycle-accurate simulators with the required level of detail for commercial architectures is a challenge, as information about precise latencies associated with, e.g., memory components, buses, or microarchitecture details are rarely available open source.

We will demonstrate the feasibility of using a non-timing-accurate simulator for effective predictions based on quantitative parameters, without relying on timing information.

Our focus is on single-core workloads. To achieve fast simulation performance, we employ gem5 in *atomic* mode with the `SimpleCPU` model for x86, RISC-V, and ARM architectures (see Section II-C). gem5 allows to replicate the cache architecture while making it parameterizable to account for cache hits, misses, and replacements within the memory hierarchy (see Section II-B).

C. Score Predictor - Workflow and Notations

To use non-timing-accurate, i.e., instruction-accurate simulators, for score prediction, a predictor must be trained. Figure 4 illustrates the training and execution phases of this predictor. During the training phase, workloads are not only executed in gem5 but also natively on the target CPU. We require a distinct predictor for each architecture and *kernel type*. For instance, Conv2D+Bias+ReLU represents one specific kernel type. Note that the corresponding predictor can be applied to any combination of shapes and parameters of this kernel type. A fixed combination of shapes and parameters for a given kernel type is referred to as a *group*. The autotuning process generates various *implementations* (schedules) for each group. In the subsequent execution phase, we leverage the pre-trained predictor. The target CPU is **not** required anymore

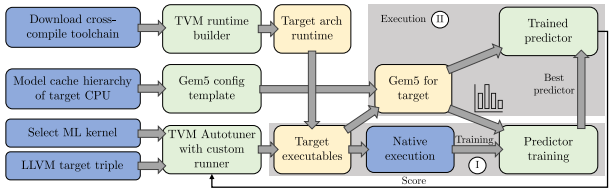


Fig. 4: Workflow of training (I) and execution (II) of a predictor for one target architecture and one kernel type

at this stage, which enables the simulation of architectures such as RISC-V on x86 platforms.

D. Score Predictor Training

The score predictor estimates a score S for an implementation I. The underlying principle is that these scores correlate with the run times, but only for different implementations of the same kernel type and group. Perfect prediction means $S_{I1} < S_{I2} \Rightarrow t_{I1} < t_{I2}$, where t_{Ix} is the run time of implementation x. Notably, comparisons between implementations across different kernel types or groups are not possible using the score. Given that timing details are unavailable, the score predictor relies solely on quantitative information rather than latencies. The relevant statistics derived from gem5 are:

- The number of the executed load/store/branch instructions divided by the total number of instructions.
- The total number of the executed instructions normalized to the total number of executed instructions of the group.
- Cache read/write replacements/hits/misses divided by read/write accesses of each cache.

Considering, e.g., L1D write ($L1Dw$) cache hits for implementation x, we determine:

$$P_{L1Dw}(I_x) = \frac{L1D_{w,hits}(I_x)}{L1D_{w,access}(I_x)} \tag{1}$$

Additionally, all parameters are also normalized to the group:

$$P_{L1Dw,norm}(I_x) = \frac{P_{L1Dw}(I_x) - \overline{P_{L1Dw}(\mathbf{I})}}{\overline{P_{L1Dw}(\mathbf{I})}} \tag{2}$$

The values $\overline{P_{L1Dw}(\mathbf{I})}$ represent the mean values across the group. We found that the most promising approach is to use these parameters as inputs for the predictor in both their original form $P_{L1Dw}(I_x)$ and their normalized form $P_{L1Dw,norm}(I_x)$. The output scores that are used for training are the measured run times normalized to the group, similar to Equation (2). Based on these inputs and the scores, we train different predictors, which can be used with different loss functions. In this work, we use Mean Squared Error (MSE), Mean Absolute Error (MAE), and Residual Sum of Squares (RSS). Below, a brief overview of the predictors is provided.

1) Multiple Linear Regression: MLR is an extension of linear regression. MLR is a simple predictor. It only models linear relationships. It finds a relation between multiple outputs y_k and multiple independent inputs x_i [20], in our case:

$$y = b_0 + b_1x_1 + b_2x_2 + ... + b_nx_n \tag{3}$$

2) Regression with DNNs: Regression using DNNs means predicting continuous outputs based on several input features. Typically, the networks consist of multiple fully connected layers, with the number of input neurons corresponding to the number of input features. Each hidden layer usually applies an activation function to introduce non-linearity and allow the network to capture complex relationships. The final layer has a number of neurons equal to the number of output variables. A common architecture might include a couple of hidden layers with decreasing numbers of neurons.

3) Bayesian Optimization: Bayesian optimization builds a surrogate model (in this case, a Gaussian process) that approximates the real/true objective function. It uses this model to predict which areas of the parameter space are likely to yield the best results and evaluates the function in those regions. An acquisition function balances the trade-off between exploring new regions of the space (exploration) and refining promising known regions (exploitation) [21]. Listing 6 shows the choice of the used objective function and loss function.

```
def objective_function(C, RBF_scale, noise):
    func = ConstantKernel(constant_value=C, ...) \
        * RBF(length_scale=RBF_scale, ...) \
        + WhiteKernel(noise_level=noise, ...)
    gp = GaussianProcessRegressor(func) \
        .fit(X_train, y_train)
    predictions = gp.predict(X_test)
    return - loss_function(y_test, predictions)
```

Listing 6: Objective function for bayes optimization

The hyperparameters of the Gaussian process are the input parameters of the `objective_function`. The Bayesian optimization framework maximizes the objective function through a series of iterations. Each iteration involves fitting a Gaussian process model with new hyperparameters and updating the probabilistic model based on the results.

4) XGBoost: Extreme Gradient Boosting (XGBoost [22]) is a powerful algorithm for regression tasks. It uses an ensemble of decision trees to predict continuous outcomes. In XGBoost, trees are built sequentially, with each tree trying to minimize the errors made by the previous ones by adjusting residuals. The algorithm optimizes a loss function using gradient descent. The hyperparameters of the XGBoost algorithm are, e.g., the learning rate, max. tree depth, and regularization parameters.

E. Score Predictor Inference

For inference, the trained predictors are integrated into the execution pipeline, as shown in Figure 4. A challenge arises because input parameters, such as $P_{L1Dw,norm}(I_x)$, cannot be determined for new, unknown groups. This limitation is due to the Auto-Scheduler generating implementations batch-wise based on prior scores (see Section II-A), preventing the computation of mean values like $\overline{P_{L1Dw}(\mathbf{I})}$ at the beginning.

To address this, we allow approximating mean values using two approaches: *static* and *dynamic* windows. In the static approach, mean values are calculated from the first $\overline{w}$ samples. In the dynamic window approach, mean values are adaptively adjusted over time. The batch size, and thus the window size $\overline{w}$, is typically large enough that no accuracy loss compared to using $\overline{P_{L1Dw}(\mathbf{I})}$ was observed in the experiments.

IV. Results

To verify the quality of our predictors, all benchmarks are executed on three different CPU architectures: x86, ARM, and RISC-V. For x86, we use a 64 bit 2.2 GHz AMD Ryzen 7 5800X 8-Core processor; for ARM, we use a 64 bit 1.5 GHz Raspberry Pi 4 Model B with an ARM Cortex-A72 processor; and for RISC-V, we use a 64 bit 1.2 GHz SiFive U74-MC processor.

TABLE I: Cache sizes and hierarchy of the used CPUs

	L1 Data			L1 Instruction			L2			LLC (L3)		
	size	sets	assoc	size	sets	assoc	size	sets	assoc	size	sets	assoc
x86	32K	64	8	32K	64	8	512K	1024	8	32768K	32768	16
ARM	32K	256	2	48K	256	3	1024K	1024	16	-	-	-
RISC-V	32K	64	8	32K	64	8	2048K	2048	16	-	-	-

TABLE II: Shapes of the used Conv2D+Bias+ReLU kernels

group	N	H	W	CO	CI	KH	KW	stride	pad
0	1	224	224	64	3	7	7	(2,2)	(3,3)
1	1	56	56	64	64	3	3	(1,1)	(1,1)
2	1	56	56	128	64	3	3	(2,2)	(1,1)
3	1	28	28	256	128	3	3	(2,2)	(1,1)
4	1	14	24	512	256	3	3	(2,2)	(1,1)

Table I lists the cache hierarchies of these CPUs. We model them in gem5 (see Section III-B). All cache line sizes are 64 B. The ARM and RISC-V CPUs feature a shared L2 cache but no L3 cache. All gem5 simulations are executed on the x86 machine. We use five groups of Conv2D+Bias+ReLU kernels from a ResNet [23] architecture as benchmarks. Table II lists the shapes and parameters of these groups. For training the predictors, the Auto-Scheduler generates 500 implementations per group, with 100 implementations used for the test set.

To determine the reference execution time t_{ref}, each implementation is executed $N_{exe} = 15$ times, with the median value taken as the reference. Additionally, cooldown times of $t_{cooldown} = 1s$ are inserted between each run to ensure more reproducible measurements. Workloads are not executed in parallel on real hardware, as this could affect the measurements. This means that execution on K parallel simulators is faster than native (sequential) execution on a single device.

$$K = \left\lceil \frac{t_{simulator}}{(t_{cooldown} + t_{ref}) \cdot N_{exe}} \right\rceil \tag{4}$$

Our measurement setup ($N_{exe} = 15$, $t_{cooldown} = 1s$) results in $K_{x86} \in [7, 97]$, $K_{ARM} \in [4, 31]$, and $K_{RISC-V} \in [3, 21]$ for the tested workloads. This means that in the best case, *only 3 parallel executions on the x86 machine are sufficient to achieve a speedup over native execution on the RISC-V CPU.*

A. Prediction of Non-Trained Groups

First, we demonstrate that a trained predictor can be utilized even when a specific kernel group was not included in the training data. This capability is crucial as we aim to develop one predictor per architecture and kernel type (e.g., Conv2D+Bias+ReLU) that remains valid across all groups with different shapes and parameters (see Section III-C).

To achieve this, we initially train Bayesian predictors for all architectures using all groups. Subsequently, we train additional predictors using only groups 0, 1, 2, 4, and 5. Figure 5 compares the test set of group 3 when group 3 is included in the training (Figures 5a to 5c) against using the same samples when group 3 is not included in the training (Figures 5d to 5f). The x-axis represents the individual samples. The reference time t_{ref} shows the median values of the measured, in ascending order sorted run times of the implementations. To obtain the predicted run time t_{pred}, the predicted scores are sorted in ascending order, with the corresponding measured run time plotted according to these scores. A perfect prediction would mean that $t_{pred} == t_{ref}$.

Although not all scores are perfectly ordered, a clear ascending trend is evident. Predictions for ARM and RISC-V architectures appear more accurate than those for x86. This discrepancy may arise from fewer HW optimizations on these embedded CPUs. Furthermore, each predictor is only as good as its reference measurements. Since execution times on x86 are significantly faster, reference measurements show greater variability compared to longer run times on the embedded architectures. Overall, visual inspections reveal no clear advantage between included and non-included training groups. *This demonstrates that the predictor can effectively be used even for groups that are not present in the training data.* To enable better comparisons of predictions, we will introduce three different evaluation metrics in the following.

B. Evaluation Metrics for the Predictors

The most important aspect is predicting the fastest run time. Therefore, we introduce E_{top1} and the rank metric R_{top1}. E_{top1} represents the relative error between the run times of the fastest reference measurement and the sample with the best predicted score:

$$E_{top1} \coloneqq \left(1 - \frac{t_{ref}[0]}{t_{pred}[0]}\right) \cdot 100\% \tag{5}$$

Rank R_{top1} indicates the relative position at which the fastest sample was ranked by the predictor:

$$R_{top1} \coloneqq \frac{100\%}{|t_{ref}|} \cdot \left(\underset{x}{\operatorname{argmin}} \left(t_{pred}[x] == t_{ref}[0]\right) + 1\right) \tag{6}$$

For example, $R_{top1} = 3\,\%$ means that the fastest sample was ranked within the top 3 % of predictions.

Additionally, we need a metric to evaluate sorting quality. Consecutive non-monotonically increasing samples should be penalized, as well as their extent of deviation. Therefore, we introduce the Quality Score Q:

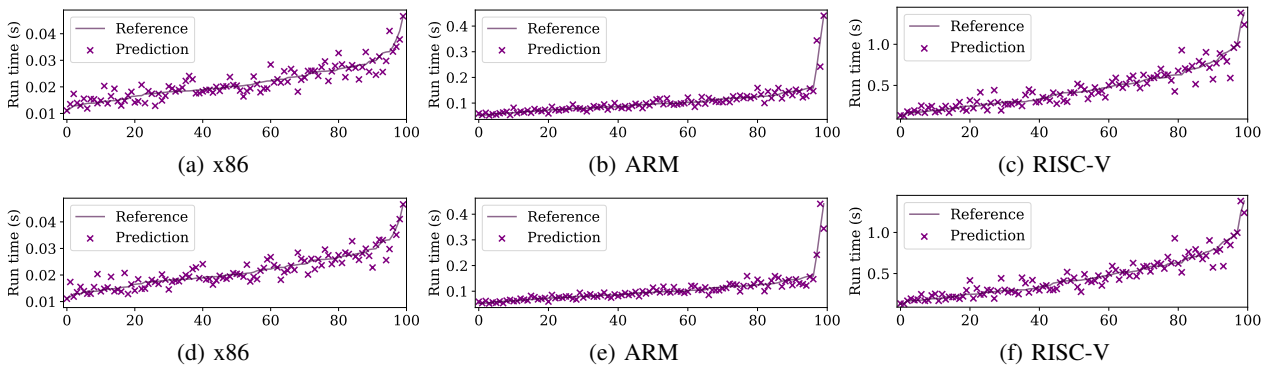
$$Q \coloneqq \frac{100\%}{|t_{ref}|} \cdot \sum_i \frac{t_{ref}[i] - min(t_{ref}[i], t_{ref}[i+1])}{t_{ref}[i]} \tag{7}$$

To prevent deviations in the slower part of the run times from being disproportionately weighted, we evaluate Q separately for the lower 50 % of the run times (Q_{low}) and the higher 50 % of the run times (Q_{high}). For all metrics, a smaller value indicates better performance.

C. Comparison of Predictors

Next, we compare the different predictors (see Section III-D). We tested various loss functions, activation functions, and parameters. Given that the XGBoost algorithm has many hyperparameters, we employed grid search [24] for tuning. The used configurations (after tuning) are:

- **Linear Regression**: RSS loss
- **DNN**: 6 dense layers (number of neurons: 128, 128, 64, 32, 16, 1), linear output activation, tanh hidden layer activation, MAE loss, adam optimizer



(a) x86 (b) ARM (c) RISC-V

(d) x86 (e) ARM (f) RISC-V

Fig. 5: Sorted run time predictions for the test set of group 3 (a)-(c) when group 3 is **included** in the training vs. the same samples of group 3 (d)-(f) when group 3 is **not included** in the training

TABLE III: Prediction results for x86-based CPU

ID	LinReg				DNN				Bayes				XGBoost			
	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)
0	10.7	3.0	3.0	8.5	1.4	2.7	2.2	3.0	12.5	3.0	2.7	3.0	7.0	2.7	2.4	2.0
1	11.2	3.6	3.5	3.0	3.2	2.8	2.8	2.0	8.9	2.6	2.8	3.5	1.5	2.7	2.4	2.5
2	15.0	3.2	3.2	3.0	7.6	2.7	2.3	2.0	0.0	2.6	2.3	1.0	2.2	2.4	2.2	2.0
3	8.8	3.2	3.0	2.0	0.7	2.4	2.4	1.5	0.7	2.7	2.7	1.5	0.0	2.8	2.4	1.0
4	0.0	3.4	3.3	1.0	2.8	3.1	2.2	3.0	5.4	2.5	2.6	2.0	2.0	2.8	2.4	2.0

- **Bayes**: Objective function shown in Listing 6, MSE loss
- **XGBoost**: Column subsample ratio 0.6, learning rate 0.05, max. tree depth 3, alpha 0, lambda 0.1, gradient boost trees 300, min. child weight 1, subsample ratio for training 0.8, MSE loss

For the evaluation, all groups were included in the training. One predictor (i.e., one of LinReg, DNN, etc.) was trained per CPU architecture. Each training was performed 10 times with a random selection of training and test sets. Scores were subsequently calculated based on the median predictions from the test sets. Table III shows the results for x86, Table IV for ARM, and Table V for RISC-V.

For the x86 architecture, the DNN, Bayesian optimization, and XGBoost yield good scores. XGBoost achieves the best average R_{top1} score, with a maximum of 2.5 %. XGBoost also had the smallest prediction error, with an average of $E_{top1} = 2.54\,\%$. For ARM and RISC-V, the scores are slightly better. For ARM, DNNs, Bayesian optimization, and XGBoost achieve a R_{top1} score smaller or equals 2.5 %. All E_{top1} errors are below 5 %, and often even below 2 %. Bayesian optimization correctly predicts the optimum in three out of five cases and demonstrates the lowest Q_{low} and Q_{high}. For RISC-V, even the linear model performs quite well, with exceptions in E_{top1} for group 0 and 4. For other predictors, the optimum is often included in the top 2 % of predictions. The DNN provides the best results with $E_{top1} \leq 3.6\,\%$ and $R_{top1} \leq 2\,\%$.

In summary, it is possible to train predictors for forecasting. *For RISC-V and ARM, a prediction error E_{top1} of less than 5 % is achievable.* If the goal is to find the best sample, it is *sufficient to re-execute the top 2 %-3 % of the predictions* later on a real architecture.

TABLE IV: Prediction results for ARM-based CPU

ID	LinReg				DNN				Bayes				XGBoost			
	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)
0	9.9	3.4	2.9	3.5	0.2	2.8	2.3	1.5	0.0	2.6	2.0	1.0	2.7	3.0	2.2	1.5
1	6.4	3.6	3.2	3.0	4.6	3.2	2.6	2.0	4.6	3.4	2.4	2.0	4.3	3.2	2.6	2.5
2	7.7	3.6	2.3	5.0	0.7	3.3	2.4	1.5	4.3	3.2	2.3	2.5	0.3	3.4	2.2	1.5
3	7.7	2.8	2.6	2.5	1.1	2.8	2.3	1.5	0.0	2.8	2.0	1.0	4.0	3.2	2.2	2.0
4	2.2	3.5	2.5	4.0	0.2	3.1	2.6	1.5	0.0	2.6	2.2	1.0	1.0	3.0	2.3	2.0

TABLE V: Prediction results for RISC-V-based CPU

ID	LinReg				DNN				Bayes				XGBoost			
	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)	E_{top1} (%)	Q_{low} (%)	Q_{high} (%)	R_{top1} (%)
0	10.9	4.0	3.9	4.0	2.2	4.0	4.0	2.0	0.0	3.8	4.2	1.0	0.0	3.4	4.1	1.0
1	0.0	4.0	4.3	1.0	0.6	3.7	4.5	1.5	2.5	3.4	4.4	2.0	4.6	3.5	4.5	2.5
2	0.0	3.4	3.7	1.0	0.0	3.2	3.8	1.0	0.0	2.8	3.4	1.0	0.0	3.0	3.8	1.0
3	0.0	3.6	3.8	1.0	0.0	3.2	3.9	1.0	2.0	2.8	3.7	1.5	4.4	3.0	3.6	2.0
4	11.0	4.0	4.2	3.0	3.6	3.6	4.2	1.5	10.7	3.2	4.0	2.0	8.2	3.8	4.0	3.0

V. Conclusion and Future Work

In this work, we introduced an interface for executing autotuning workloads on simulators and explored the feasibility of using instruction-accurate simulators for autotuning of ML workloads. We trained and compared different score predictors for the x86, ARM, and RISC-V architectures. Our results show that the tuned predictors can identify optimal implementations within the top 3 % of predictions. In the case of limited availability of the target HW, our approach can even be used to accelerate autotuning. In the best case, 3 parallel simulations on the used x86 machine were sufficient to replace one RISC-V board. Our research lays the groundwork for using instruction-accurate simulations for performance estimation.

Future work will focus on benchmarking a broader range of CPUs to train more generalized predictors. These generalized predictors can then be applied to previously untested CPUs, enhancing our methodology's appeal for pre-silicon software development.

REFERENCES

[1] K. Datta *et al.*, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," in *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE, 2008, pp. 1–12.

[2] L. Lin and M. Gen, "Auto-tuning strategy for evolutionary algorithms: balancing between exploration and exploitation," *Soft Computing*, vol. 13, pp. 157–168, 2009.

[3] T. Chen *et al.*, "TVM: An automated End-to-End optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[4] L. Zheng *et al.*, "Ansor: Generating High-Performance tensor programs for deep learning," in *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, 2020, pp. 863–879.

[5] Y.-S. Hsieh and Y.-P. You, "DLOOPT: An Optimization Assistant on AutoTVM for Deep Learning Operators," *Journal of Signal Processing Systems*, vol. 95, no. 5, pp. 585–607, 2023.

[6] W. Grunewald and T. Ungerer, "Towards extremely fast context switching in a block-multithreaded processor," in *Proceedings of EUROMICRO 96. 22nd Euromicro Conference. Beyond 2000: Hardware and Software Design Strategies*. IEEE, 1996, pp. 592–599.

[7] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of scheduling techniques for addressing shared resources in multicore processors," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–28, 2012.

[8] T. Benoit-Cattin, D. Velasco-Montero, and J. Fernández-Berni, "Impact of thermal throttling on long-term visual inference in a CPU-based edge device," *Electronics*, vol. 9, no. 12, p. 2106, 2020.

[9] D. Brodowski, N. Golde, R. J. Wysocki, and V. Kumar, "CPU frequency and voltage scaling code in the Linux (TM) kernel," *Linux kernel documentation*, vol. 66, 2013.

[10] F. Bellard, "QEMU, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41, no. 46. California, USA, 2005, pp. 10–5555.

[11] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[12] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[13] J. Du and Y. Xu, "Hierarchical deep neural network for multivariate regression," *Pattern Recognition*, vol. 63, pp. 149–157, 2017.

[14] P. I. Frazier, "Bayesian optimization," in *Recent advances in optimization and modeling of contemporary problems*. Informs, 2018, pp. 255–278.

[15] T. Chen *et al.*, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[16] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *Acm Sigplan Notices*, vol. 48, no. 6, pp. 519–530, 2013.

[17] V. Kelefouras and G. Keramidas, "Design and implementation of deep learning 2D convolutions on modern CPUs," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[18] R. Li, Y. Xu, A. Sukumaran-Rajam, A. Rountev, and P. Sadayappan, "Analytical characterization and design space exploration for optimization of CNNs," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 928–942.

[19] S. Kumar and P. Singh, "An overview of modern cache memory and performance analysis of replacement policies," in *2016 IEEE International Conference on Engineering and Technology (ICETECH)*. IEEE, 2016, pp. 210–214.

[20] G. K. Uyanık and N. Güler, "A study on multiple linear regression analysis," *Procedia-Social and Behavioral Sciences*, vol. 106, pp. 234–240, 2013.

[21] P. I. Frazier, "A tutorial on Bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[22] T. Chen, T. He, M. Benesty, and V. Khotilovich, "Package 'xgboost'," *R version*, vol. 90, no. 1-66, p. 40, 2019.

[23] Z. Wu, C. Shen, and A. Van Den Hengel, "Wider or deeper: Revisiting the ResNet model for visual recognition," *Pattern recognition*, vol. 90, pp. 119–133, 2019.

[24] S. Putatunda and K. Rama, "A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost," in *Proceedings of the 2018 international conference on signal processing and machine learning*, 2018, pp. 6–10.

Exploiting Power Side-Channel Vulnerabilities in XGBoost Accelerator

Yimeng Xiao, Archit Gajjar, Aydin Aysu and Paul Franzon
Dept. Electrical and Computer Engineering
North Carolina State University
Raleigh, NC, USA
{yxiao32, amgajjar, aaysu, paulf}@ncsu.edu

***Abstract*—XGBoost (eXtreme Gradient Boosting), a widely-used decision tree algorithm, plays a crucial role in applications such as ransomware and fraud detection. While its performance is well-established, its security against model extraction on hardware platforms like Field Programmable Gate Arrays (FPGAs) has not been fully explored. In this paper, we demonstrate a significant vulnerability where sensitive model data can be leaked from an XGBoost implementation through side-channel attacks (SCAs). By analyzing variations in power consumption, we show how an attacker can infer node features within the XGBoost model, leading to the extraction of critical data. We conduct an experiment using the XGBoost accelerator FAXID on the Sakura-X platform, demonstrating a method to deduce model decisions by monitoring power consumptions. The results show that on average 367k tests are sufficient to leak sensitive values. Our findings underscore the need for improved hardware and algorithmic protections to safeguard machine learning models from these types of attacks.**

***Index Terms*—XGBoost, hardware security, side-channel attack, HLS, FPGAs.**

I. Introduction

The increasing reliance on machine learning (ML) applications has driven the development of hardware accelerators, designed to meet growing demands for computational performance and energy efficiency [1]. FPGAs have become a popular choice in edge computing, where they accelerate critical ML tasks such as training and inference. However, these environments inherently process sensitive data, such as proprietary models and personal information, raising significant security concerns. Ensuring the confidentiality of these data is paramount, especially given the threat of SCAs, which exploit non-functional signals like power consumption to extract private information [2]–[9].

SCAs have been successfully employed to reveal sensitive components of neural networks (NNs), including hyperparameters, model weights, and input data [10]–[13]. Such attacks pose a critical security risk, as they have been demonstrated to reverse-engineer models or reconstruct sensitive inputs from architectures like Binary Neural Networks (BNNs) [14] and Convolutional Neural Networks (CNNs) [15]. Despite the focus on NNs [16], less attention has been given to tree-based models like XGBoost, which are increasingly deployed in security-sensitive environments.

Tree-based models, such as the Bonsai tree based classifiers, are known to be vulnerable to SCAs, including correlation power analysis (CPA) and template attacks (TA) [17]. XGBoost, however, differs from simple decision tree structure due to its iterative boosting process. Therefore, extending such attacks to XGBoost is non-trivial and unexplored. In this paper, we present a novel power-based SCA specifically designed to exploit the vulnerabilities of XGBoost, with a focus on environments where sensitive data is processed under strict resource constraints.

We demonstrate the vulnerability of the FAXID [18] to power-based SCAs, focusing on the extraction of critical model components such as node features. By carefully analyzing power consumption patterns during the model's decision-making process, we establish a strong correlation between power consumptions and the traversal of specific branches in the decision trees. This enables an attacker to reconstruct the entire sequence of decisions, revealing the model's internal logic and exposing sensitive data. Such vulnerabilities could allow attackers to bypass defenses and compromise both the model and underlying data.

Our key contributions are as follows:

- We introduce a novel SCA that leverages power consumption patterns to extract sensitive data as node features, from XGBoost models.
- We successfully demonstrate the attack on the FAXID, providing detailed insights into the security vulnerabilities.
- Our study reveals that even with the higher abstraction level provided by High-Level Synthesis (HLS), critical model information, such as node features, remains vulnerable to SCAs.

The paper is organized as follows: Section 2 outlines the threat model and presents background information on our attack methodology. Section 3 explains our rationale for using SCAs to extract data from FAXID. Section 4 presents experimental results, including statistical analyses of the leaked data patterns, and discusses potential defense strategies. Finally, Section 5 concludes the paper.

II. Background

In this section, we discuss the threat model of our attack and provide essential background knowledge on several key topics, including decision trees, HLS, SCAs, and power analysis.

A. HLS

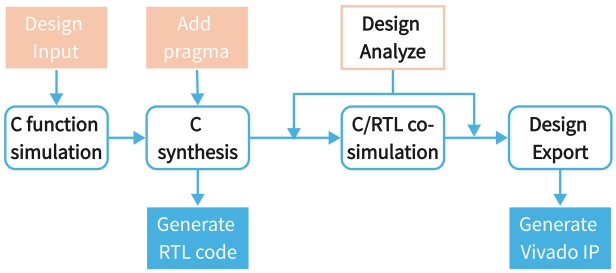


Fig. 1: The HLS design flow treats C functions as hardware modules, with pragmas guiding the hardware configuration within the code. The flow culminates in generating a Vivado IP block that integrates smoothly with RTL code.

HLS is a design process that is used to design and specify digital hardware systems using high-level programming language, e.g. C, C++, or System C. After simulation and synthesis, the software tool such as AMD/Xilinx Vitis will generate hardware description language (HDL) code from C/C++ code. Then, the C code will run a co-simulation. The final step is to generate a Vivado Intellectual Property (IP). Figure 1, the design flow for HLS, shows how to use high-level language to design hardware.

The key benefit of HLS is that it allows hardware engineers to design and implement hardware functionality without having to manually write the detailed low-level hardware descriptions typically required in traditional hardware design. This can significantly speed up the development of custom hardware for specific applications and make it more accessible to software developers who may not have extensive hardware design expertise.

HLS brings numerous advantages and diverse applications, yet security within HLS-based designs remains a significant concern. Numerous research efforts have been undertaken to investigate the security aspects of HLS design [19] and the secure deployment of ML models on hardware platforms [11], [13]–[15], [17]. Existing SCAs on HLS [19] have predominantly focused on cryptographic applications, examining how memory-based optimizations in HLS influence power side-channel leakage. However, discussions regarding the effects of architectural design and data flow on power side-channel leakage in the context of model extraction of ML classifiers are notably absent in the existing body of work.

B. XGBoost

XGBoost is a scalable and efficient machine learning algorithm known for its strong performance in tasks such as classification, regression, and ranking [20]. The model begins by evaluating input data at the root node, where the feature determines the direction of traversal through the tree, continuing until a leaf node is reached. Unlike traditional methods, XGBoost uses a sigmoid function at the leaf node to provide probability-based outputs, improving prediction granularity [18]. Node features are crucial, as they dictate the traversal path and influence decisions. Figure 2 illustrates an example where input data is first evaluated based on gender, then age, ultimately leading to a final classification (e.g., identifying an individual as a father if age is below 50).

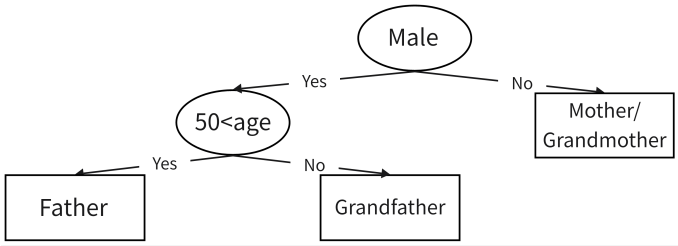


Fig. 2: A simple decision tree generated by XGBoost on a small dataset. The tree splits based on age and gender, classifying individuals into family roles such as Father, Grandfather, Mother, or Grandmother.

C. SCA

SCAs exploit information unintentionally leaked by computer systems or cryptographic devices, not by targeting the cryptographic algorithm itself [21], but by analyzing external channels such as power consumption, electromagnetic emissions, timing variations, and even acoustic signals [22]. By monitoring these signals, attackers can deduce sensitive information, such as encryption keys or private data, thus compromising system security. Power consumption is a key side channel in SCAs [23], as a device's power usage fluctuates depending on the operations performed. By analyzing these power variations, attackers can infer internal states and extract critical information, making power side-channel analysis a highly effective method for undermining system security.

D. Power Analysis

Power analysis is a critical tool in research for determining the minimum sample size needed to distinguish between two distinct datasets following Gaussian distributions [24]. This analysis hinges on three essential parameters: effect size, significance level, and statistical power. Effect size quantifies the magnitude of a result within the population using measures. The significance level sets the threshold for accepting or rejecting the null hypothesis. Meanwhile, statistical power indicates the probability of correctly accepting the alternative hypothesis if true.

E. Threat Model

Our threat model follows the typical assumption in SCAs, specifically in power-based model extraction in ML-based systems [17], [25]. In this paper, we hypothesize that the attacker has physical access to a device employing XGBoost for ransomware detection. Consequently, the adversary is capable of running multiple tests by applying inputs and capturing corresponding power consumption traces. Additionally, we assume that the attacker has an in-depth understanding of both the operational mechanics and the architectural framework of the FAXID [18]. This expertise enables the construction of power models for individual branches, facilitating effective

```
1  function get_leaf_single(current_sample, column,
       data, tree, leaf):
2      for depth = 0 to MAX_DEPTH:
3          if tree[node].is_leaf:
4              leaf = tree[node].leaf_value
5              break
6          else:
7              if data[current_sample * column + tree
                   [node].feature] < tree[node].
                   split:
8                  *node = tree[node].left
9              else:
10                 *node = tree[node].right
```

Fig. 3: Pseudo code illustrating the vulnerable `get_leaf_single` subroutine in XGBoost. The branching logic on lines 8 and 10, determining node traversal, leads to distinguishable power consumption patterns, making it susceptible to SCAs.

differentiation between them. In the scenario described, the attacker's objective is to locate the feature data stored within the FAXID during the calculation of branch decisions for each tree node.

III. The Proposed SCA

In this section, we analyze the vulnerability of the FAXID [18] to SCAs, focusing on the distinct risks it faces. We begin by comparing our context with the previous attack on the implementation of the Bonsai algorithm [17], discussing the structural differences between the models. Next, we identify the key components in FAXID that are most susceptible to SCAs and investigate how these weaknesses manifest at the hardware level. Finally, we demonstrate how these attacks can compromise sensitive information, emphasizing the importance of securing FAXID against SCAs.

A. Comparison with Previous Work on Bonsai Algorithm

In prior research [17], the focus has been on the Bonsai algorithm, which is specifically designed for decision-making in IIoT systems using decision trees. The Bonsai algorithm operates by employing a function that processes input data to calculate branch choices at each decision point. This makes the function parameters critical to the decision-making process, rendering them vulnerable to SCA. Attackers can exploit these parameters to disrupt the tree structure by identifying sensitive information.

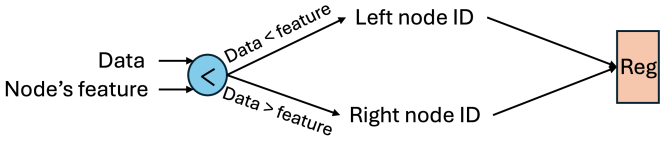


Fig. 4: The input data and node feature determine which subnode ID is stored in the register. With the feature fixed, varying the input data influences the resulting node ID. Different node IDs typically result in distinct power consumption patterns, enabling a correlation between input data and power consumption.

Our research presents a novel method to attack FAXID, motivated by several significant differences from the Bonsai algorithm:

- **Distinct Function Mechanisms**: Unlike Bonsai, our target algorithm does not use a decision-making function that processes input data to determine branch choices. Instead, XGBoost utilizes two distinct calculation functions:
  - **Comparison Calculation**: This function determines the branch by comparing the input data with a node feature.
  - **Sigmoid Function**: This function uses weak probabilities from all the trees to calculate the final probability, indicating the likelihood of the data being classified as ransomware.
- **Lack of Specific Parameters**: The most critical difference is that in these two functions, there are no specific parameters that can be guessed or manipulated. While Bonsai's function parameters are essential for its decision-making process and thus susceptible to attacks, XGBoost's comparison and sigmoid calculations do not expose similar parameter-based vulnerabilities.

These fundamental differences necessitate a new approach for effectively attacking the FAXID. Our method addresses these distinctions to explore novel avenues of exploitation, demonstrating the need for a targeted strategy that is distinct from those employed against the Bonsai algorithm.

B. Exfiltrating Models through SCAs

The FAXID system [18] employs a key computational function, `get_leaf_single`, which is designed for efficient tree traversal and retrieval of the leaf node. This function plays a critical role in FAXID's ability to quickly identify and extract the leaf node, where the decision or value based on input features is stored, as shown in Figure 3.

Figure 3 also illustrates a pseudo code snippet that is vulnerable to SCAs. The vulnerability lies in the fundamental comparison operation during binary decision-making, where the input data is compared to the node feature. If the input data is less than the feature, the process moves to the left node; otherwise, it branches to the right, providing an entry point for attackers to exploit power consumption patterns.

Our strategic approach is designed to accurately locate the node feature for each node within a binary tree structure. This goal is achieved through a two-step process. Firstly, a binary search is conducted to determine the range of the node feature for the branches. This range is progressively narrowed down through further binary searches, allowing us to pinpoint the exact node features.

This strategy enables efficient identification of node features in a binary tree using power analysis to estimate the minimum sample size needed to distinguish between branches, enhancing attack efficiency [26]. After comparing the input data with the node feature, the corresponding node ID is stored for subsequent calculations, as shown in Figure 4. By monitoring

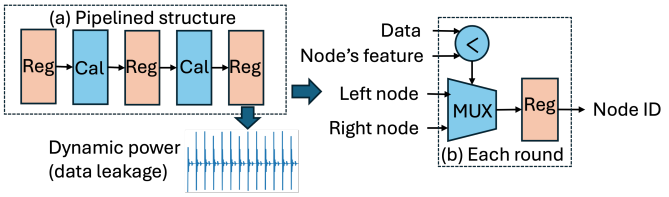


Fig. 5: A hardware implementation of the `get_leaf_single` function: (a) shows the pipelined structure, highlighting the significant power consumption differences when storing values to the register; (b) details the calculation process during each iteration.

power consumption patterns and adjusting input data, we can uncover the tree's dynamics, leading to precise and accurate identification of node features [6].

C. Assessing Structural Vulnerabilities

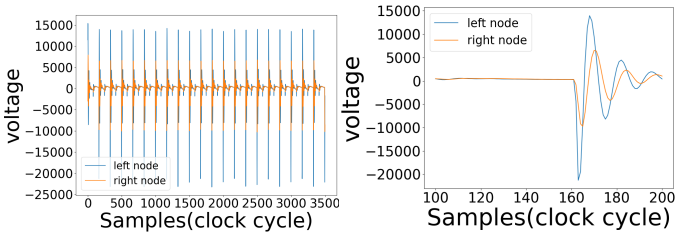
The conversion of complex algorithms into a HDL through HLS results in the formation of hardware IP, which is subsequently integrated seamlessly onto an FPGA board. Figure 5 depicts the post-implementation configuration of the `get_leaf_single` function on the FPGA. Figure 5 (a) unveils a pipeline structure generated by HLS to enhance processing throughput, while Figure 5 (b) offers a detailed view of the computational framework highlighted earlier.

Each branch typically exhibits two distinct power consumption profiles when there are differences in the Hamming distances of node IDs: one for nodes with a higher Hamming distance and another for those with a lower one. By altering input data and observing subsequent shifts in power consumption, the traversal path can often be inferred—revealing whether the data is progressing toward a left or right node. However, in some cases, subnodes may have the same Hamming distance, making it difficult to distinguish between them through power consumption alone. As branching continues in subsequent rounds, new subnodes often exhibit different Hamming distances, resulting in distinct power consumption patterns. An attacker can exploit these evolving power traces across rounds to differentiate nodes.

This scenario presents a significant security risk: unintended leakage of information about the node features in an XGBoost model due to power consumption variations during data operations. By analyzing large volumes of data and monitoring these power variations, attackers can infer the feature range of nodes with increasing precision. Notably, this attack does not require compromising all nodes simultaneously. Exploiting a single node allows the attacker to apply the same technique iteratively, thereby revealing the conditions of subsequent nodes and progressively compromising the entire model.

IV. Experiment Results

In this section, we first describe the experimental setup, providing an overview of the hardware and software components used. We then outline the statistical methods employed to analyze the patterns of leaked data and evaluate the success rate of the SCA. Our findings reveal that data leakage persists even with minor variations in Hamming distance. Additionally,



(a) Power Profiles (b) Close-up of (a)

Fig. 6: The power consumption varies based on branch selection. (a) displays power profiles for left and right node selections, showing distinct patterns across clock cycles. (b) provides a close-up of (a), highlighting the noticeable difference in voltage spikes between the two branch selections.

the results show that neither the data type nor the proximity between the input data and the node feature significantly affects the occurrence of data leakage. Finally, we discuss potential countermeasures to mitigate the impact of this attack, offering strategies to enhance system security.

A. Experiment Setup

We successfully scaled our code from the U50 FPGA to the Sakura-X platform, enabling efficient side-channel testing. Using Xilinx Vitis, we generated core IP from HLS code, while Xilinx Vivado was employed to integrate peripheral circuits with the IP. The Sakura-X, operating at a stable frequency of 24 MHz, provided a reliable environment for testing. To improve the clarity of power traces, we used a 20dB amplifier, ensuring more precise measurements. For data acquisition, a PicoScope 3206D high-resolution oscilloscope was configured with a sampling rate of 250 MS/s to capture detailed power consumption data throughout the experiments.

B. Feasibility Check

Before extracting Points of Interest (POIs), we first assess whether the power traces of the two branches differ enough to separate them effectively during an attack. To maximize the Hamming distance and demonstrate the feasibility of our SCA, we assigned an ID of 0 to the right node and 255 to the left node, creating the greatest power difference for easier detection.

Experiments were conducted with a constant node feature of 2.5, using input values of 1 and 4 to generate power traces for both branches. An average of 1,000 power traces was collected to ensure accurate profiling. Figure 6 (a) shows the resulting power profiles for the left and right nodes, with clear differences, particularly when zoomed in, as illustrated in Figure 6 (b). These variations enable precise identification of the selected node based on power consumption.

C. POI Analysis

In our analysis, POIs are defined by power consumption patterns associated with subnode IDs stored in a register. A key challenge is the variability in power peaks during computations, which requires an understanding of their fluctuation range. To address this, we employ Kernel Density Estimation

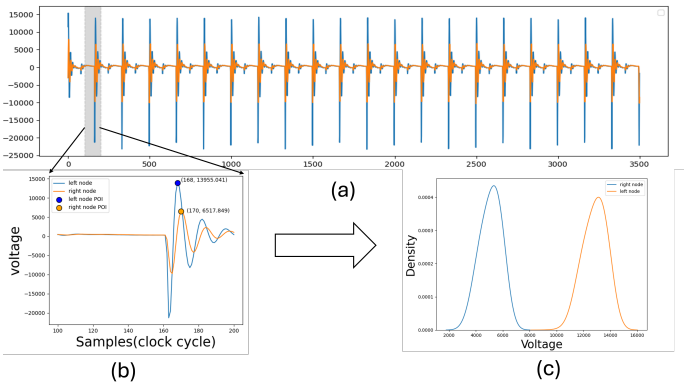


Fig. 7: Extraction of POI from Power Traces and Corresponding Power Distribution: (a) shows two overlapping power traces across clock cycles, (b) zooms in to highlight the identified POI in both traces, and (c) displays the voltage distribution at the POI based on 1000 traces, clearly separating the power profiles of the left and right nodes.

(KDE) [27], [28], a data smoothing technique used to derive the Probability Density Function (PDF) [29], which maps power consumption density at POIs. Utilizing the PDF allows us to estimate the probability of power consumption falling within a specific range, providing insight into the distribution of power across POIs.

To analyze the impact of power distribution on correcting our attack, we focus on the first POI. Figure 7 illustrates the process of extracting this POI from power traces, which involves identifying peak values for power distribution analysis. The procedure begins by determining the ranges of the POIs within the power traces. Subsequently, the first range is isolated. Following this, the peak value within this range is identified, which is crucial for constructing the power distribution. Moreover, we use 1,000 data points per node to calculate the power distribution, ensuring a robust analysis.

Figure 8 illustrates three distinct scenarios related to POIs, represented by peak voltage on the x-axis and distribution percentage on the y-axis. These scenarios highlight the impact of distribution overlap on subnode identification as the Hamming distance between two node IDs decreases. As the Hamming distance narrows, the POI distributions begin to overlap, eventually fully converging when the distance becomes zero. In this context, Figure 8 summarizes the three key cases, demonstrating the progressive overlap of distributions as the node IDs become increasingly similar.

The analysis highlights that reduced overlap in power distribution correlates directly with increased accuracy in attack correction. Subnode IDs are typically arranged sequentially, resulting in clearer differentiation in power distributions, as seen in cases similar to Figures 8 (a) and 8 (b). However, as discussed in Section 3C, when the power distributions at the POI for two nodes fully overlap (Figure 8 (c)), distinguishing between them based on power consumption becomes difficult. This challenge is confined to the current round of operation. In subsequent rounds, nodes branch into subnodes with different

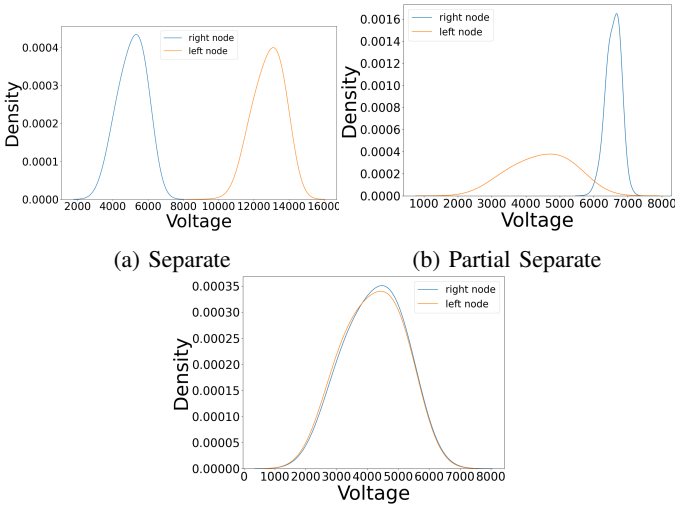


Fig. 8: Three power distributions are shown at POI. (a) The distributions are fully separate, clearly indicating the branch choice based on power. (b) The distributions are partially separated, making branch distinction less reliable. (c) The distributions fully overlap, making it impossible to differentiate branch choices from power measurements.

Hamming distances, leading to distinct power patterns. Attackers can leverage these evolving patterns by analyzing power traces over multiple rounds to differentiate the nodes. While full POI overlap complicates initial attacks, subsequent rounds provide new opportunities for node identification through the behavior of subnodes.

D. Sample Size Analysis

In the previous section, we abstracted the POIs from the power traces, allowing us to accurately pinpoint the differences in the power traces. Using this information, power analysis helps us determine the minimum sample size needed to separate branches effectively. While calculating POIs, the sample size is consistently set at 1000. However, Figures 8 (b) and 8 (c) may require more samples for a clear separation, whereas Figure 8 (a) might need fewer samples. For our power analysis, we set the significance level (α) at 0.05 and the statistical power (β) at 80%. The effect size is calculated using Cohen's d.

Figure 9 presents the decision tree used in our analysis, comprising 28 nodes, including 14 decision nodes, each branching into two paths. For these decision nodes, we calculated the minimum sample size required to reliably distinguish between branches. Notably, as illustrated in Figure 10, we observed an inverse relationship between the difference between the input data and node features and the required sample size. Specifically, input data farther from node features typically requires fewer samples to separate subnodes, as larger differences lead to greater variations in power consumption when processing the comparison, thereby enhancing the ease of differentiation.

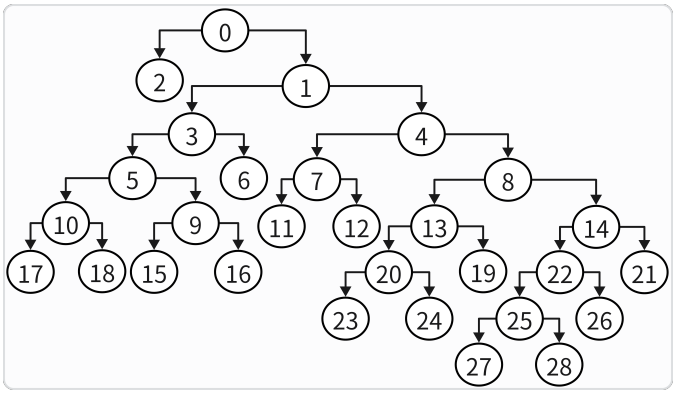


Fig. 9: The decision tree used in our experiment, where each node is labeled with its corresponding index, illustrating the structure and node numbering used for branching decisions.

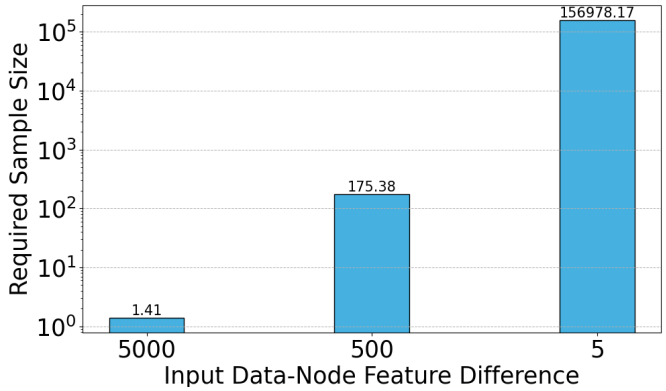


Fig. 10: The required sample size increases as the difference between the input data and the node feature decreases. A smaller difference of 5 requires 156,978 samples, while larger differences, such as 500 and 5000, require only 175 and 1.41 samples, respectively.

Figure 11 provides a detailed breakdown of the sample sizes for the 14 decision nodes, excluding leaf nodes. These decision nodes are fundamental in defining the tree's structure and play a critical role in assessing classification boundaries. The required sample size for each node is influenced by the range of features associated with that node, with a maximum of 32 attempts per node due to floating-point precision constraints. On average, 367,218 samples are required per node to fully compromise the tree. In the worst case, a node demands 5,023,301 samples, while in the best case, only 65 samples are needed.

E. Potential Countermeasures

To mitigate node ID-based attacks, two potential solutions can be considered, both aimed at eliminating the influence of the Hamming distance on power consumption. The first approach involves modifying the node IDs to ensure that, regardless of which ID is stored in the register, the Hamming distance between IDs remains uniform. This can be achieved without altering the tree structure itself, by simply assigning IDs in such a way that they have a consistent

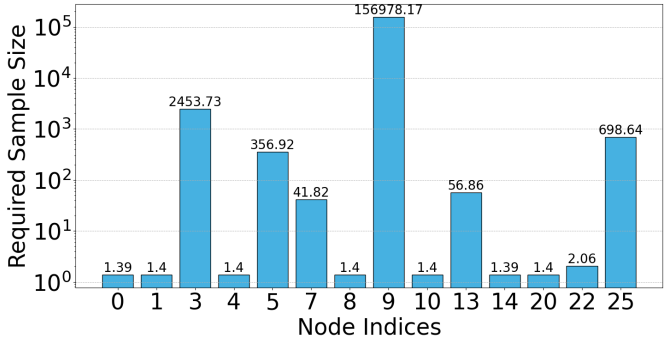


Fig. 11: The required sample size for distinguishing each node. The bar plot illustrates the minimum number of samples needed to differentiate between node decisions based on power consumption reliably.

Hamming distance. This method effectively prevents attackers from using variations in power consumption to differentiate between nodes, while preserving the integrity and performance of the tree. However, the drawback is the added complexity of managing ID assignments to maintain consistency.

A second approach to mitigating SCAs is to modify hardware by integrating mechanisms that equalize power consumption, independent of the Hamming distance between node IDs [30]–[33]. This involves adding circuitry to ensure uniform power usage, masking variations that could be exploited by attackers. While effective, this method increases complexity and costs, potentially affecting performance, especially in power-constrained environments like edge computing. Alternatively, software-based masking techniques [34]–[36] randomize power consumption to hinder correlation with operations, though both approaches introduce trade-offs in terms of performance or efficiency, as the tree structure may become less optimized for typical machine learning tasks.

V. Conclusion

In this paper, we analyzed the vulnerability of FAXID to SCAs. Our results demonstrate that power consumption during execution can be exploited to infer branch decisions and identify node features through a binary search technique. In our experiments, breaking a single node in a decision tree with 28 nodes, including 14 branches, required an average of 367,000 samples. These findings underscore the importance of implementing robust defense mechanisms, such as hardware equalization or masking techniques, to mitigate the risk of SCAs in FPGA-accelerated ML models.

VI. Acknowledgements

This research is based upon work supported by the National Science Foundation under the Grants No. CNS - 2137283 – Center for Advanced Electronics through Machine Learning (CAEML) and its industry members.

References

[1] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Comput. Surv.*, vol. 48, no. 4, mar 2016. [Online]. Available: https://doi.org/10.1145/2893356

[2] S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic, and N. Mentens, "Side-channel Analysis and Machine Learning: A Practical Perspective," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 4095–4102.

[3] S. Wolf, H. Hu, R. Cooley, and M. Borowczak, "Stealing Machine Learning Parameters via Side Channel Power Attacks," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021, pp. 242–247.

[4] X. Yan, X. Lou, G. Xu, H. Qiu, S. Guo, C. H. Chang, and T. Zhang, "MERCURY: An Automated Remote Side-channel Attack to Nvidia Deep Learning Accelerator," in *2023 International Conference on Field Programmable Technology (ICFPT)*, 2023, pp. 188–197.

[5] S. HajiAmin Shirazi, H. Naghibijouybari, and N. Abu-Ghazaleh, "Securing Machine Learning Architectures and Systems," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 499–506. [Online]. Available: https://doi.org/10.1145/3386263.3409104

[6] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI Neural Network: Using Side-channels to Recover Your Artificial Neural Network Information," Cryptology ePrint Archive, Paper 2018/477, 2018. [Online]. Available: https://eprint.iacr.org/2018/477

[7] E. Debenedetti, G. Severi, N. Carlini, C. A. Choquette-Choo, M. Jagielski, M. Nasr, E. Wallace, and F. Tramèr, "Privacy Side Channels in Machine Learning Systems," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 6861–6848.

[8] J. Liu and H. Wang, "Model Extraction Attack against On-device Deep Learning with Power Side Channel," in *2024 25th International Symposium on Quality Electronic Design (ISQED)*, 2024, pp. 1–5.

[9] Q. Pan, J. Wu, A. K. Bashir, J. Li, and J. Wu, "Side-Channel Fuzzy Analysis-Based AI Model Extraction Attack With Information-Theoretic Perspective in Intelligent IoT," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 11, pp. 4642–4656, 2022.

[10] H. Weerasena and P. Mishra, "Revealing CNN Architectures via Side-Channel Analysis in Dataflow-based Inference Accelerators," *ACM Trans. Embed. Comput. Syst.*, vol. 23, no. 6, sep 2024. [Online]. Available: https://doi.org/10.1145/3688001

[11] H. Chabanne, J. Danger, L. Guiga, and U. Kühne, "Side Channel Attacks for Architecture Extraction of Neural Networks," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 1, p. 3–16, Mar 2021.

[12] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. A. Faruque, "Stealing Neural Network Structure Through Remote FPGA Side-Channel Analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4377–4388, 2021.

[13] N. Gupta, A. Jati, and A. Chattopadhyay, "AI Attacks AI: Recovering Neural Network architecture from NVDLA using AI-assisted Side Channel Attack," Cryptology ePrint Archive, Paper 2023/368, 2023, https://eprint.iacr.org/2023/368. [Online]. Available: https://eprint.iacr.org/2023/368

[14] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Power Side-Channel Attacks on BNN Accelerators in Remote FPGAs," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 357–370, 2021.

[15] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I Know What You See: Power Side-Channel Attack on Convolutional Neural Network Accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 393–406. [Online]. Available: https://doi.org/10.1145/3274694.3274696

[16] S. Maji, U. Banerjee, and A. P. Chandrakasan, "Leaky Nets: Recovering Embedded Neural Network Models and Inputs Through Simple Power and Timing Side-Channels—Attacks and Defenses," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12 079–12 092, 2021.

[17] D. Jap, V. Yli-Mäyry, A. Ito, R. Ueno, S. Bhasin, and N. Homma, "Practical Side-Channel Based Model Extraction Attack on Tree-Based Machine Learning Algorithm," in *Applied Cryptography and Network Security Workshops*, J. Zhou, M. Conti, C. M. Ahmed, M. H. Au, L. Batina, Z. Li, J. Lin, E. Losiouk, B. Luo, S. Majumdar, W. Meng, M. Ochoa, S. Picek, G. Portokalidis, C. Wang, and K. Zhang, Eds. Cham: Springer International Publishing, 2020, pp. 93–105.

[18] A. Gajjar, P. Kashyap, A. Aysu, P. Franzon, S. Dey, and C. Cheng, "FAXID: FPGA-Accelerated XGBoost Inference for Data Centers using HLS," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–9.

[19] L. Zhang, D. Mu, W. Hu, Y. Tai, J. Blackstone, and R. Kastner, "Memory-Based High-Level Synthesis Optimizations Security Exploration on the Power Side-Channel," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2124–2137, 2020.

[20] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[21] A. Pulkit, S. Naval, and V. Laxmi, "A Survey of Side-Channel Attacks in Context of Cache – Taxonomies, Analysis and Mitigation," 2023. [Online]. Available: https://arxiv.org/abs/2312.11094

[22] P. Wang, S. Nagaraja, A. Bourquard, H. Gao, and J. Yan, "SoK: Acoustic Side Channels," 2023. [Online]. Available: https://arxiv.org/abs/2308.03806

[23] A. Ghimire, V. V. Baligodugula, and F. Amsaad, "Power Analysis Side-Channel Attacks on Same and Cross-Device Settings: A Survey of Machine Learning Techniques," in *Internet of Things. Advances in Information and Communication Technology*, D. Puthal, S. Mohanty, and B.-Y. Choi, Eds. Cham: Springer Nature Switzerland, 2024, pp. 357–367.

[24] C. Lu, E. Telang, A. Aysu, and K. Basu, "Quantum Leak: Timing Side-Channel Attacks on Cloud-Based Quantum Services," 2024. [Online]. Available: https://arxiv.org/abs/2401.01521

[25] F. Aydin and A. Aysu, "Exposing Side-Channel Leakage of SEAL Homomorphic Encryption Library," in *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security*, ser. ASHES'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 95–100. [Online]. Available: https://doi.org/10.1145/3560834.3563833

[26] E. Cagli, C. Dumas, and E. Prouff, "Convolutional Neural Networks with Data Augmentation against Jitter-Based Countermeasures – Profiling Attacks without Pre-Processing," Cryptology ePrint Archive, Paper 2017/740, 2017. [Online]. Available: https://eprint.iacr.org/2017/740

[27] Y.-C. Chen, "A Tutorial on Kernel Density Estimation and Recent Advances," 2017.

[28] T. C. Bailey, A. C. Gatrell *et al.*, *Interactive spatial data analysis*. Longman Scientific & Technical Essex, 1995, vol. 413, no. 8.

[29] C. M. Grinstead and J. L. Snell, *Introduction to probability*. American Mathematical Soc., 1997.

[30] M. Isakov, L. Bu, H. Cheng, and M. A. Kinsy, "Preventing Neural Network Model Exfiltration in Machine Learning Hardware Accelerators," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2018, pp. 62–67.

[31] A. Dubey, R. Cammarota, V. Suresh, and A. Aysu, "Guarding Machine Learning Hardware Against Physical Side-channel Attacks," *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, apr 2022. [Online]. Available: https://doi.org/10.1145/3465377

[32] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "GuardNN: Secure Accelerator Architecture for Privacy-Preserving Deep Learning," 2022. [Online]. Available: https://arxiv.org/abs/2008.11632

[33] X. Yan, C. H. Chang, and T. Zhang, "Defense against ML-based Power Side-channel Attacks on DNN Accelerators with Adversarial Attacks," 2023. [Online]. Available: https://arxiv.org/abs/2312.04035

[34] A. Dubey, R. Cammarota, and A. Aysu, "MaskedNet: The First Hardware Inference Engine Aiming Power Side-Channel Protection," 2019. [Online]. Available: https://arxiv.org/abs/1910.13063

[35] A. Dubey *et al.*, "ModuloNET: Neural Networks Meet Modular Arithmetic for Efficient Hardware Masking," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, pp. 506–556, Nov. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9306

[36] A. Dubey, R. Cammarota, and A. Aysu, "BoMaNet: Boolean Masking of an Entire Neural Network," 2020. [Online]. Available: https://arxiv.org/abs/2006.09532

Centralized Training and Decentralized Control through the Actor-Critic Paradigm for Highly Optimized Multicores

Benedikt Dietrich, Heba Khdr, Jörg Henkel
Chair for Embedded Systems, Karlsruhe Institute of Technology (KIT), Karlsuhe, Germany
benedikt.dietrich@kit.edu, heba.khdr@kit.edu, henkel@kit.edu

***Abstract*—While distributed, neural-network-based resource controllers represent the state of the art for their ability to cope with the ever-expanding decision space, such approaches suffer from several limitations, like conflicting control decisions and partial observability. These effects can significantly impair the controllers' learning capabilities and the stability of their control policies, causing substantial performance losses. We are the first to solve this problem employing a centralized training and decentralized control regime to mitigate the aforementioned limitations. Specifically, we design a centralized neural network (critic) that evaluates the behavior of multiple decentralized neural controllers (actors) in a system-wide context. The objective of our proposed technique is to maximize the performance under a temperature constraint through dynamic voltage frequency scaling. The evaluation of our technique shows its superiority over the state of the art, yielding average (peak) performance improvements of 20% (34%), which we consider a breakthrough as the gains are measured on a real-world platform.**

I. Introduction

Runtime performance optimization is essential to get the most out of modern processors while adhering to system-imposed resource constraints, such as power budgets or temperature limits. Merely relying on design-time knowledge is increasingly insufficient for deriving efficient resource control policies that perform well in every possible runtime situation. Therefore, research shifted focus to online learning techniques. Reinforcement learning is particularly suited for adapative resource control due to its two key properties [1].

1) It has the capability to find complex control strategies when it is challenging to define them analytically, provided only with the objective formulated as a reward signal. By framing resource control as a reinforcement learning problem, the underlying processor can be effectively treated as a black box, obviating the requirement for exhaustive design-time knowledge about the processor and the running applications.
2) It allows for continual adaptation to changing environment conditions by constantly monitoring the environment's feedback to its decisions and accordingly updating the parameters of the control policy. This property is especially important in an open system, where workload combinations are virtually limitless, requiring frequent adjustments to the resource allocation between the running applications to maintain optimal performance.

Reinforcement learning has been proposed to implement centralized controllers, focusing primarily on coarse-grained control levers [2], [3], such as power capping or system-wide dynamic voltage and frequency scaling (DVFS), or operating within small solution spaces involving few cores and control decisions [4], [5]. However, centralized control is impractical for systems with multiple concurrently running applications and different workload characteristics, as the solution space grows exponentially with the number of cores and control knobs, slowing down convergence and incurring unacceptable runtime overhead. To enable scalable, finer-grained control, decision-making can be distributed among multiple lower-level controllers each focusing on a sub-system of the processor (e.g., a core or a cluster of cores). Distributed control introduces its own set of challenges [6]. With multiple controllers acting concurrently, it becomes increasingly difficult for each of them to trace the impact of its own decisions on the processor's state and system-wide objectives. This is known as the *credit assignment* problem in the reinforcement learning literature. Furthermore, each local controller has only a partial view of the entire system. Although this reduces the complexity of the learning problem, acting under *partial observability* also means that the controller can lack the complete information necessary to identify the optimal decision for the current situation. An additional challenge compared to the centralized setting is that the environment becomes increasingly *non-stationary*, as the same action can lead to different outcomes at different times as other controllers adjust their policies concurrently.

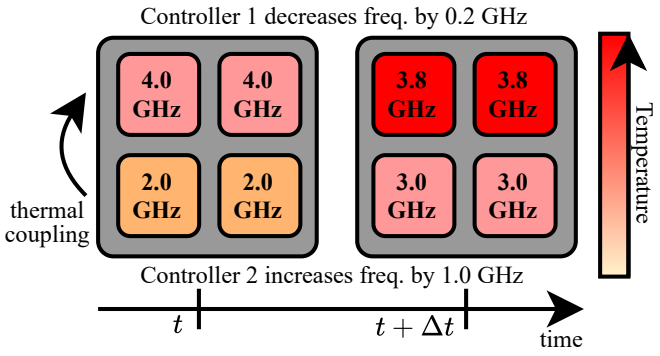


Fig. 1. Example to illustrate potential interference between controllers. Both controllers are assumed to operate independent of each other. They can only observe the cores under their supervision. However, the control problems are not indepedent and concurrent influence on the shared environment makes it challenging for each controller to trace the impact of its own actions. In this concrete example, the situation may arise that controller 1 observes an increase in temperature even if it has decreased its frequency due to a simultaneous frequency increase by controller 2 and thermal coupling between the cores.

The following example shows these challenges in the context of processor resource control. We focus on the task of DVFS for temperature control. One controller is assigned to each running application and sets the frequency for the associated cores. Figure 1 illustrates how these changes affect an example processor with four cores. The initial frequency assignments are shown on the left, with controller 1 running its application at 4 GHz and controller 2 at 2 GHz. Since higher frequencies lead to higher core temperatures, controller 1 may decide to reduce its frequency by 0.2 GHz. Controller 2 measures that its current core temperature justifies an increase in frequency. After applying the new frequencies, the controllers measure the resulting temperatures in the next step. Since the control problems are interdependent due to thermal coupling among processor cores, the situation may arise where, despite controller 1 *decreasing* its frequency, it observes an *increase* in the temperature of its own cores.

Such counterfactual observations are typical for uncoordinated control, resulting in controller interference [7] and sub-optimal and/or unfair resource distributions [8]. To address these issues, prior research has explored hierarchical and self-organizing coordination schemes for distributed resource control [9], [10]. The approach in [9] is representative of a hierarchical technique in which a central arbiter configures the cost function of each local controller. It focuses on power-constrained performance optimization using per-core DVFS and distributes the system's power budget proportional to each core's utilization. However, this heuristic is insufficient for other, more complex objectives like maximizing system performance while adhering to a temperature constraint (due to spatial and temporal thermal effects). In [10], the goal is to optimize performance under a temperature constraint in a many-core processor. It uses one neural controller per core that tries to maximize the core's performance while keeping its temperature below the threshold. The state of each local controller is extended to include the temperature of its neighboring cores. Although this approach broadens the view of each controller, it does not fully resolve the challenges outlined above, as the controllers pursue individual performance objectives.

This paper is the first to approach resource control through *centralized training and decentralized execution* [6]. This paradigm allows us to address the inherent challenges of distributed resource control while still retaining the benefit of parallelizable decision-making and simplified learning problems of each controller. Specifically, we build on the *actor-critic paradigm* that distributes decision-making among multiple neural controllers, known as *actors*, and learns to evaluate the joint resource control using a second network, the *critic*. During centralized training, individual actors (controllers) share their local experiences with the critic, enabling it to evaluate each controller's decision-making in a global context. As a result, the actors can update their policies based on the critic's evaluation, aligning their decision-making with the *shared objective*: optimizing the overall system performance within the processor's limited resources. We apply this paradigm to the task of temperature control in a multicore processor within an open-system context, where various unknown applications may arrive to execute concurrently on the processor, competing for the thermal budget.

In summary, this work offers the following **novel contributions**.

- We propose a centralized training and decentralized execution approach for multicore resource control.
- Following this approach, we introduce an actor-critic-based technique to maximize the overall system performance under a temperature constraint through DVFS, while continuously adapting to varying workloads.
- We provide a lightweight implementation of our temperature control and assess its performance on a real processor, showcasing its ability to improve the performance within an open-system context, where a priori unknown applications are executing concurrently and competing for shared resources.
- We measure superior results compared to the state of the art on this real-world platform.

II. Related Work

We give an overview of related work on (distributed) resource control. In addition, we provide relevant background information on cooperative reinforcement learning.

A. Multicore Resource Control

Multicore resource control has been a long-standing area of research. State-of-the-art resource control uses machine learning to enable application-aware decision-making, even for a priori unknown applications. For example, in a recent DVFS technique, the authors of [11] use a long short-term memory network to classify application phases into known workload categories and apply the optimal frequency for each category. Although models trained at design time can generalize to unseen applications, it is virtually impossible to develop one that performs well across all potential workloads, environmental conditions, and device configurations likely to occur during operation. Reinforcement learning provides a promising alternative, enabling end-to-end learning of control decisions based on current performance counter and sensor measurements and adaptation to the aforementioned runtime dynamics. Recent works leverage deep reinforcement learning techniques, including value-based methods [10], [12] and policy-gradient methods [13]–[15] to implement control policies. In parallel, both centralized approaches with a single controller and decentralized approaches with multiple controllers have been proposed in the literature. Centralized approaches tend to focus on coarse-grained control levers [2], [3] or small solution spaces [4], [5]. To enable more fine-grained control, decision-making can be distributed among multiple lower-level controllers, each dedicated to a subsystem of the processor. Uncoordinated decision-making by these local controllers can result in suboptimal resource allocation and even destructive interference [7], [8]. To address these challenges, prior work has explored both self-organizing [10], [16] and hierarchical coordination schemes [9], [17]. Self-organizing systems rely on direct communication among agents, while hierarchical schemes decompose the global objective into local performance targets and resource constraints for each low-level controller. Some hierarchical approaches propose centralized searches in the joint action space of controllers [7], [18], [19]. However, such techniques require calling a central arbiter in every control interval, adding an additional step to the latency-critical control loop. Alternatively, custom resource distribution schemes have been developed for specific tasks, such as power-constrained performance optimization [9], [20]. These task-specific schemes are not easily transferable to other objectives, and deriving a decomposition of system objectives into local objectives remains challenging for dynamic objectives such as temperature control. Although prior work has explored cooperative decentralized control or neural controllers independently, no existing approach has fully integrated cooperative decentralized control and (deep) neural networks to optimize resource control. So far, the literature has only presented trials that use multiple neural controllers for scalability and to manage complexity, but none of them has addressed resource control with a dedicated cooperative reinforcement learning algorithm. In this work, we propose a resource control framework based on a state-of-the-art cooperative reinforcement learning algorithm [21], following the principle of *centralized training for decentralized execution*.

B. Background: Cooperative Control with Reinforcement Learning

Reinforcement learning involves an agent interacting with an environment (observing the state of the environment s and taking actions a to influence it) in order to carry out a certain task, formulated as the maximization of a scalar feeback signal (reward r) [1]. Distributed resource control can be formulated as a cooperative reinforcement learning problem, where multiple agents, placed in a shared environment, need to collaborate to achieve a shared objective (maximize a shared reward signal). The objective could be, for example, to minimize the system's energy-delay product or maximize the average application throughput while maintaining power and thermal constraints. Typically, agents perceive a local subset o_i of the state of the underlying environment s [6]. The agents follow a policy $\pi(a|o_i)$ that maps these local observations to actions. Nevertheless, the environment is shared among all agents, so that both the shared reward and the environment's state (and in turn the local observations) are influenced by the joint behavior of all agents. Cen-

tralized training and decentralized execution is a popular paradigm to enable cooperation among agents [6]. Under this paradigm, agents act independently but collaborate for the optimization of their control policies. The actor-critic family of algorithms aligns naturally with this paradigm [6]. In centralized training and decentralized execution under the actor-critic paradigm, there are multiple actors (neural networks that learn the local policies) and a single centralized critic (a neural that learns to evaluate them). Each policy is conditioned on its local observation $\pi(a|o_i, \theta_i)$ and parameterized by a set of parameters θ_i. The central critic is trained as a regression model with a dedicated set of parameters ϕ and predicts the expected reward $Q(s, \mathbf{a}, \phi)$ under the joint behavior of the actors $\mathbf{a} = \{a_1 ... a_n\}$ and conditioned on the underlying system state s. At a high level, the critic evaluates actions based on their influence on the achievable reward, while the actors adjust their policies to increase (decrease) the likelihood of selecting actions that yield high (low) rewards.

III. Centralized Training and Decentralized Control for Highly Optimized Multicores

We propose a cooperative resource control framework based on centralized training and decentralized execution following the actor-critic paradigm. Prior research has primarily focused on learning neural control policies, often paired with rule-based, task-specific resource allocation schemes. We argue that learning how to optimally assign resources to controllers is just as important as training the local controllers, as it opens the way for a more powerful control that supports complex, system-wide objectives, expressed as a shared reward signal, without in-depth prior knowledge about the processor and applications. To this end, we implement a state-of-the-art actor-critic algorithm [21] that uses neural networks to learn both the control policies (through the actor networks) and the resource allocation strategies (through the centralized critic network) at runtime. A major advantage of our approach is that the local controllers can execute in parallel without the need to communicate with other controllers or with a central arbiter to arrive at the final control decision. The control decisions are evaluated *in hindsight* (outside the latency critical control loop) during centralized training, when the controllers share their recent experiences (state observations and actions) with the critic and receive feedback about their control decision, allowing cooperative coordination of the system's resources. To illustrate how our approach works in practice, we implement a technique that follows the aforementioned scheme of centralized training and decentralized control to solve the problem of performance maximization under a temperature constraint using DVFS.

A. Problem Formulation

We focus on a processor with N_c cores with per-core clock signals and N_f discrete frequency levels. The goal is to find a frequency allocation across cores that maximizes the average throughput among concurrently running applications while making sure that the maximum core temperature remains below a critical threshold T_{crit}. We measure performance as the inverse of execution time: t_{exe}^{-1}. Multiple, multi-threaded applications execute in parallel and contend for the processor's thermal budget. We consider an open system, where the applications and their arrival times are considered unknown at design time. We assign incoming applications to the first free cores[1] and each core executes one thread of the application. The initial mapping remains fixed throughout the application's execution.

[1]Our DVFS control is orthogonal to mapping decisions; it can be combined by different mapping policies.

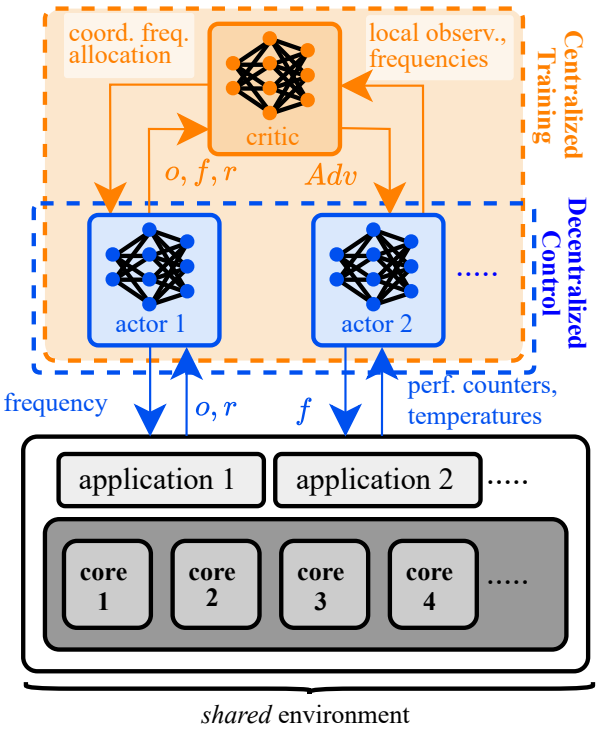


Fig. 2. Decentralized temperature control incorporating the actor-critic paradigm. Local actors implement the control policies, managing the core frequencies of an associated application. The actors engage in centralized training to coordinate their frequency decisions to maximize the average system performance under the processor's thermal budget. The coordination is implicitly performed by the central critic which evaluates the controllers' local frequency decisions within the system-wide context and with regard to the shared performance objective and temperature constraint. The actors align their decision to the frequency allocation recommended by the critic.

The problem can be summarized as follows.

objective: max. system perf. (avg. t_{exe}^{-1} across apps.)
constraint: $\max_c T_c \leq T_{crit}$
means: per-core DVFS

B. Our Approach using the Actor-Critic Paradigm

We propose using an actor-critic algorithm to solve this optimization problem. Each application is managed by a dedicated controller (actor) that adjusts the frequency of the cores associated with the application. We opt for agents at the application level, as they have knowledge about different threads of a multi-threaded application. Furthermore, it ensures that all threads of the application run at the same speed, preventing stalls at synchronization points [22]. To achieve the objective outlined above, we propose a shared reward signal that consilidates the throughputs of the different applications and enforces a system-wide temperature constraint. Due to the spatial and temporal heat transfer, system-wide thermal control cannot be easily decomposed into local, independent sub-problems for each controller. Such a decomposition would require detailed information about the processor's thermal behavior (e.g., floorplan, device node) and cooling system. In our approach, the *central critic* evaluates the frequency allocation considering the interactions of all local DVFS controllers with the processor. Using the representational power of neural networks, the critic learns the complex interdependencies between core activity, performance, and thermal behavior. An overview of our approach is provided in Fig. 2. It is described in detail in the next sections.

1) State, Reward, and Actions: We first focus on the local controllers and formalize the state and action spaces, as well as the shared reward signal.

State (Observation): Each controller can consider only a fraction of the underlying processor state to derive its control decisions. We use the following features in the observation of each controller. The frequency, average instructions per cycle (IPC), last-level cache misses, and utilization across the cores under the controller's supervision during the previous execution interval, as well as their current average and maximum temperature. These features characterize the performance and thermal behavior of the application. The impact of a frequency change is highly application-specific. Furthermore, different execution phases of the same application require different frequency selection. In general, compute-intensive application phases benefit from higher frequencies. During memory-intensive phases (characterized by many misses on the last-level cache), computation happens faster than new data arrive at the processor, and consequently, increasing the frequency will not convert into performance gains. At the same time, a higher frequency always leads to increased dynamic power consumption ($P_{dynamic} \sim f^3$) and consequently higher temperature. Therefore, considering these application characteristics is crucial for an adaptive frequency selection that balances potential performance gains and increasing temperature.

(Shared) Reward: Temperature control requires balancing potential performance gains between multiple applications while maintaining thermally-safe operation. We propose a shared reward signal that consolidates these two objectives. The first part of the reward is based on a shared performance metric, which uses the geometric mean of the applications' instructions per second (IPS) as a surrogate for overall system performance Eq. (1).

$$r_t = \begin{cases} \textit{violation-penalty} & \exists\ c \in \{1...N_c\} \text{ s.t. } T^c_{t+1} > T_{crit} \\ \left(\prod_{i=1}^{N_a} ips^i_{t+1}\right)^{\frac{1}{N_a}} & \text{otherwise} \end{cases} \quad (1)$$

Here, N_a refers to the number of applications currently running. This metric encourages agents to adjust their frequencies in a way that improves the collective performance, without favoring any particular application. It also promotes fairness by tending toward a more balanced performance distribution across applications: The geometric mean increases as the performance deviation among the applications decreases. The second part of the reward penalizes all agents when the temperature constraint is exceeded on any core of the system. This is crucial, as thermal violations can result from frequency increases on other cores (by other controllers) and/or at earlier points in time, due to spatial and temporal thermal effects. By penalizing all agents, the reward emphasizes the entangled nature of the problem.

Actions: The controllers select from the entire set of discrete frequency levels exposed by the processor, thereby allowing fine-grained and opportunistic DVFS. Each frequency has an associated voltage level that is automatically set when the frequency is changed.

2) Decentralized Control: During the decentralized execution phase, the actors are periodically invoked within the control loop (see Algorithm 1) and execute in parallel relying only on their local observations. The number of active controllers depends on the utilization of the system. The controllers log their experiences into a local buffer, maintaining a trajectory of the most recent interactions $(o_{i,t}, f_{i,t})$. In practice, the reward is not directly emitted by the environment. It suffices to *compute* reward in hindsight during the centralized training phase, combining the observations of all agents. Therefore, the controllers also store the IPS and the maximum temperature among their cores. When the trajectory reaches a predefined length, one round of centralized training is initiated.

Algorithm 1 Decentralized Temperature Control

```
for each actor i do                                   ▷ in parallel
    while local buffer ≤ max-traj-length do
        acquire local state o_i
        sample next frequency f_i ~ π(f_i|o_i, θ_i)
        set core frequency f_i
        ...                                           ▷ application process
        monitor ips_i, T_i^max
        store (o_i, f_i, ips_i, T_i^max) in local buffer
optimize critic φ and local actors {θ_i}
repeat with updated actors
```

3) Centralized Training: Centralized training involves training the centralized critic and then updating control policies [21]. The optimization of the critic is described in Algorithm 2. First, the shared reward is calculated from the local IPS and peak temperature measurements according to Eq. (1). This is done for every time step within the trajectory. Afterward, the state of the central critic is composed. The critic considers the local trajectories of all controllers and their selected actions [21]. We compute a central state for each controller which is the concatenation of its own observation o_i and the mean of the observations of the concurrently running controllers o_{-i}. Furthermore, we add the maximum f^{max}_{-i} and mean f^{avg}_{-i} frequency selections of the concurrently running controllers and the number of active applications (controllers) N_a.

$$s_i = o_i \cup o_{-i} \cup \{f^{max}_{-i}, f^{avg}_{-i}, N_a\} \quad (2)$$

The critic state is computed for every controller and for every timestep in the trajectory. It is fed through the critic network to provide an estimate of the expected reward (called value) for each of the agent's actions given its own observation o_i plus the system-wide context $o_{-i} \cup \{f^{max}_{-i}, f^{avg}_{-i}, N_a\}$. Since all value predictions are derived with the same network ϕ, they can be computed in a batch over the length of the trajectory and the number of actors. The $TD(\lambda)$ algorithm [1] and a delayed target network ϕ to recursively compute regression targets, labeled Y^γ, for the optimization of the critic network.

$$Y^\lambda_{i,t} = r_t + \gamma \cdot \Big((1-\lambda) \cdot Q(s_{i,t+1}, f_{i,t+1}, \phi')) + \lambda \cdot Y^\lambda_{i,t+1}\Big) \quad (3)$$

The target value at the final time step of the trajectory is set to $Y^\gamma_T = Q(s_{i,T}, f_{i,T}, \phi')$. We show the temporal index here because it is relevant for recursive computation. We implicitly assume that the steps outlined in the algorithms need to be executed for all time steps, and therefore, for reasons of readability, do not show the temporal index there. The critic's parameters are updated using gradient descent on the mean-squared error loss, completing the optimization of the critic Eq. (4)).

$$\mathcal{L}_Q = (Q(s, f, \phi)) - Y^\lambda)^2 \quad (4)$$

The loss is computed over across all timesteps in the trajectory and all actors. This completes the optimization of the central critic.

The reward estimates of the critic are used to derive the feedback for each controller's frequency selection, which is optimized in the second step of centralized training (see Algorithm 3). This feedback is called the action's advantage. It is the difference between the critic's value prediction and an action-independent baseline. The purpose of this subtraction is to reduce the variance of the subsequent update of the control policies [1]. To compute the advantage term, we build on the counterfactual baseline introduced in [21] (see Eq. (5)).

$$Adv(s_i, f_i, \phi) = Q(f_i, s_i, \phi) - \sum_{j=1}^{N_f} \pi(f_j, o_i, \theta) \cdot Q(f_j, s_i, \phi) \quad (5)$$

Algorithm 2 Centralized Training (Critic Network)

for each time step in traj. **do** ▷ batch processing
  compute reward r from local ips_i,T_i^{max} ▷ Eq. (1)
  for each actor i **do** ▷ batch processing
    compute critic states s_i ▷ Eq. (2)
    compute critic values $Q(s_i, f_i, \phi)$
    compute target critic values $Q(s_i, f_i, \phi')$
for each time step in traj **do** ▷ sequential processing
  for each actor i **do** ▷ batch processing
    compute $TD(\lambda)$ targets Y^{λ} ▷ Eq. (3)
compute critic loss $\mathcal{L}_Q$ ▷ Eq. (4), (mean over traj. & actors)
update critic params. $\phi \leftarrow \phi + \alpha \cdot \frac{\partial \mathcal{L}_Q}{\partial \phi}$
update target critic params. $\phi' \leftarrow \tau \cdot \phi + (1-\tau) \cdot \phi'$

Algorithm 3 Centralized Training (Actor Networks)

for each step in traj. **do** ▷ batch processing
  for each actor i **do** ▷ batch processing
    compute critic values $Q(s_i, f_i)$
for each actor i **do** ▷ in parallel
  for each step in traj **do** ▷ batch processing
    compute freq. distr. $\pi(f|o_i, \theta_i)$
    compute advs. $Adv(o_i, f_i)$ ▷ Eq. (5)
  compute $\mathcal{L}_{\pi,i}$ ▷ Eq. (6), mean over traj.
  update actor params. $\theta_i \leftarrow \theta_i - \alpha_\pi \cdot \frac{\partial \mathcal{L}_{\pi,i}}{\partial \theta_i}$

The counterfactual advantage measures the difference of the return of the selected frequency and the average return assuming that all other controllers keep their current frequency selection $\{f_{-i}^{max}, f_{-i}^{avg}\}$. It answers the question: "Given the concurrent workload, how does a controller's frequency decision compare to all other possible frequencies?" therefore facilitating credit assignment (see Fig. 1).

To optimize the control policies $\pi(a|o_i, \phi)$, the advantage term is used as part of the loss function (see Eq. (6)), guiding the actors toward frequencies with high predicted reward while avoiding those with low predicted reward.

$$\mathcal{L}_\pi = -log(\pi(f_i, o_i, \theta_i)) \cdot Adv(s_i, f_i, \phi) - c_H \cdot H(\pi(f|o_i, \theta_i)) \quad (6)$$

Here, $\pi(f_i, o_i, \theta_i)$ is the probability of the frequency that was actually chosen by controller i, while $\pi(f_i|o_i, \theta_i)$ is the probability distribution conditioned on the observation o_i. The entropy term $c_H \cdot H(\pi(f|o_i, \theta)$ with parameter c_H is used to balance exploration and exploitation [23], [24]. The loss is averaged over the entire trajectory.

4) Configuration & Parameters: The local controllers are invoked every 10 ms. Both the actor and the critic network use two layers with 64 neurons. We use the $tanh$ non-linearity. The learning rate of the critic (0.001) is set one order of magnitude higher than the one of the actors (0.0001) [25]. The maximum trajectory length is set to 128, resulting in one round of centralized training every 1.3 s. We set the violation penalty to -100 and scale the reward to be in the range of -1 and 1 [26]. We use a discount factor of 0.9. For the target update rate τ and for λ, we follow the recommendations from the literature [21], [25]. We set the entropy coefficient to 0.001 and normalize the state vectors of the actor and critic using running estimates of the mean and variance for each state feature [26]. In addition, we use weight sharing between actors, i.e., $\theta = \theta_1 = ... = \theta_n$ [21].

IV. Evaluation

To evaluate our framework, we implement the actor-critic architecture on a real Intel® Core™ i9-12900KF processor [27], which features eight performance cores with individual clock lines and 44 frequency levels ranging from 0.8 GHz to 5.1 GHz. We implement

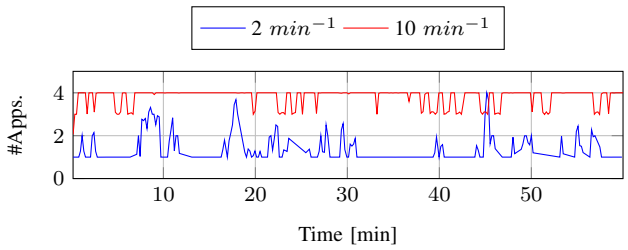


Fig. 3. Number of concurrently running applications demonstrating the varying degrees of system utilization to which the actors and the critic need to adapt. Higher utilization leads to more contention for the thermal budget, while low utilization allows individual applications to opportunistically maximize their performance. Data recorded for two and ten applications per minute under our technique.

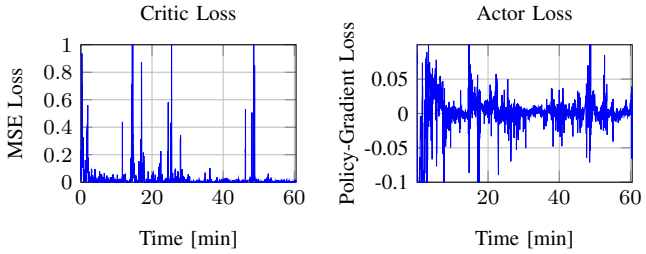


Fig. 4. Loss curves for the critic and actor networks during the first hour of the evaluation. The critic learns faster since it needs to estimate the actors' current behavior [31]. Short spikes in the critic loss are likely caused by previously unseen workload combinations. The loss quickly goes back to a low value indicating the critic's capability to adapt. Data recorded for the arrival rate of six applications per minute.

a Linux kernel module to read the performance counters and set the core frequencies. A userspace program implements the actor and critic algorithms and uses *libtorch* [28] for neural network training and inference.

A. Evaluation Workloads and Scenarios

We evaluate our approach using the following applications from the benchmark suites *PARSEC* [29] and *SPLASH-2* [30]: *barnes*, *fft*, *fluidanimate*, *lu*, *lu* (n.c.), *ocean*, *ocean* (n.c.), *swaptions*, *radiosity*, and *radix*. This workload constitutes a mixture of diverse application characteristics, with varying execution phases, peak performance, and maximum temperature. To test the techniques under varying system utilization levels (and consequently varying degrees of contention for the thermal budget), we conduct evaluation runs with five different application arrival rates: two, four, six, eight, and ten arriving applications per minute. To maximize workload diversity across the eight cores, each application uses two threads. This setup results in frequent changes to the optimal frequency allocation between applications, requiring the controllers to adapt their frequency selection dynamically (see Fig. 3). For all experiments, we use a temperature constraint T_{crit} of $75°C$. To fairly compare the approaches, we run them for two hours, giving them enough time to learn the characteristics of the environment. We report the evaluation results for the first hour separately from the second hour. During the first hour, the controllers explore various frequency selections, and the critic learns which frequency allocations are beneficial for thermally-safe and performant operation. Fig. 4 shows the loss curve for our critic and actor networks during this phase, exemplarily for a scenario with six applications per minute. In the second hour, the controllers apply the learned policies while still being able to adapt to unseen workload combinations.

B. Comparision Baselines

1) TP-DQL [10]: It uses one neural controller per core to maximize application performance under a thermal constraint. It is the

closest technique to our work, as it targets the same problem and uses neural controllers. The controllers share the same neural network and use deep Q-learning with a shared replay buffer, to enable cooperation through knowledge sharing. The reward signal is individual to each core, and each agent tries to maximize the core frequency while keeping the core temperature just below the threshold.

2) Profit [9]: It is based on tabular Q-learning. It is designed for power-constrained performance optimization and uses per-core controllers that each train an individual policy. A central coordinator distributes local power budgets to each local controller according to the cores' IPC. This heuristic for deriving local constraints is not applicable for temperature control. Therefore, we change the local power constraint to a fixed temperature constraint. We use the same penalty value as in our reward function.

C. Evaluation Results

Figure 5 shows the results of our evaluation, in terms of the execution time of applications, the IPS, and also the number of thermal violations. Our method increases overall performance across applications by 20% and 34% (average/maximum) compared to *TP-DQL*, and by 24% and 65% (average/maximum) when compared to *Profit*. As the application arrival rate increases, so does the average execution time, due to increased contention for the thermal budget. In such cases, controllers must resort to lower frequencies to keep the temperature within the critical threshold. Importantly, the room for performance optimization is significantly reduced. That explains why all comparison techniques show similar performance at the arrival rate of ten applications per minute, at which the utilization of the processor reaches 100%. All techniques manage to keep the average temperature below the critical threshold, with our approach closing the thermal margin to the threshold, effectively using the chip's thermal budget. Looking at the thermal violation results, one can observe that all comparison techniques manage to avoid thermal violations in the exploitation phase. For our technique, the maximum violation during the exploitation phase is equal to 0.05 violations per control epoch, which is minimal. However, during the exploration phase, where the agents are supposed to select random actions (frequency selections) to learn the consequence of these actions, we notice that the agents have tested high frequencies, resulting in some violations (0.2 violations per control epoch) in the scenario where system utilization reaches 100 %. Experiencing such scenarios lets the actors learn to avoid high frequencies at high system utilization to not violate the temperature constraint. Therefore, we can directly see that in the exploitation phase, our approach experiences almost no violations, demonstrating the ability to adapt to the high system utilization.

The improvement in overall system performance, both with regard to response time and throughput, while maintaining a comparable rate of thermal violations, highlights the superiority of our approach over the state of the art.

D. Overhead Discussion

Our resource control runs on the same cores as the applications. The main contributors to the overhead of our approach is the inference overhead as part of the control loop and the optimization of the actor and the critic networks during centralized training. Inference of the actor network(s) and sampling the next frequency on average takes 565 μs in each control epoch, i.e., 10ms, in our evaluation. Centralized training occurs every 1.3 s and takes 20 ms on average. In summary, the inference overhead is 5.7 % and the optimization overhead is 1.6 %. The overhead of the entire control loop, including reading the performance counters and associated context switches, as well as the overhead of the optimization, is already considered in our evaluation results, since we compute the performance improvements using the execution time which comprises the application runtime and the overhead of the techniques. Our technique stands out with a light memory footprint, as it only needs to store the most recent experiences for each controller. Compared to *TP-DQL*, which relies on experience replay and stores 100,000 samples, the memory overhead of our technique is 99 % lower. The total size of all buffers never exceeds 20 kB.

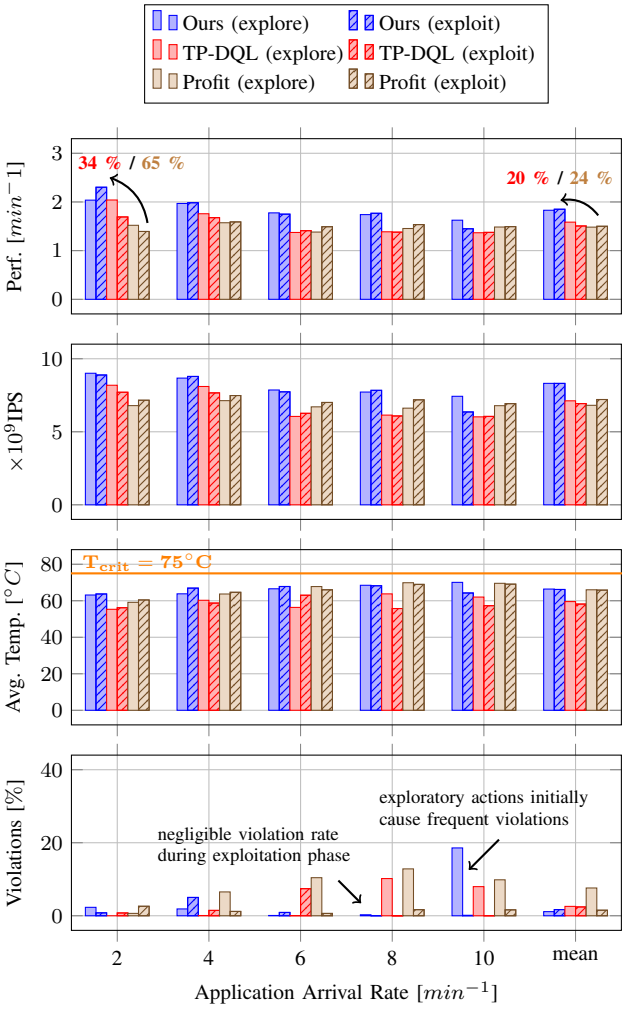


Fig. 5. Results of our evaluation with regard to performance (t_{exe}^{-1}), throughput, temperature and the percentage of thermal violations throughout the execution.

V. Conclusion

Deep reinforcement learning is a promising approach to processor resource control. Deployment to real hardware requires multiple controllers to handle the vast solution space involved. However, distributed decision-making introduces additional challenges, such as interference between concurrent controllers and partial observability, that have not been sufficiently addressed by prior resource control techniques. This work addressed these challenges using a state-of-the-art actor-critic algorithm and achieves cooperative temperature control while maintaining the scalability of distributed decision-making. We have solved an outstanding problem, and we report significant performance gains that represent from now on the state of the art for the problem of thermal-aware multicore computing.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Y. Wang, W. Zhang, M. Hao, and Z. Wang, "Online power management for multi-cores: A reinforcement learning based approach," *IEEE Trans. Parallel Distributed Syst.*, vol. 33, no. 4, pp. 751–764, 2022.

[3] J. Zhang, G. Yu, Z. He, L. Ai, and P. Chen, "Deeppower: Deep reinforcement learning based power management for latency critical applications in multi-core systems," in *52nd International Conference on Parallel Processing, ICPP*. ACM, 2023, pp. 327–336.

[4] S. Yang, Y. Wang, D. Liu, X. Jiang, H. Fang, Y. Yang, and M. Zhao, "Releta: Reinforcement learning for thermal-aware task allocation on multicore," *CoRR*, vol. abs/1912.00189, 2019.

[5] D. Liu, S. Yang, Z. He, M. Zhao, and W. Liu, "CARTAD: compiler-assisted reinforcement learning for thermal-aware task scheduling and DVFS on multicores," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 6, pp. 1813–1826, 2022.

[6] A. Wong, T. Bäck, A. V. Kononova, and A. Plaat, "Deep multiagent reinforcement learning: challenges and directions," *Artif. Intell. Rev.*, vol. 56, no. 6, pp. 5023–5056, 2023.

[7] R. Bitirgen, E. Ipek, and J. F. Martínez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *41st Annual IEEE/ACM International Symposium on Microarchitecture MICRO*. IEEE Computer Society, 2008, pp. 318–329.

[8] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "ARTE: an application-specific run-time management framework for multi-core systems," in *IEEE 9th Symposium on Application Specific Processors, SASP*. IEEE Computer Society, 2011, pp. 86–93.

[9] Z. Chen, D. Stamoulis, and D. Marculescu, "Profit: Priority and power/performance optimization for many-core systems," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 37, no. 10, pp. 2064–2075, 2018.

[10] H. Khdr, M. E. Batur, K. Zhou, M. B. Sikal, and J. Henkel, "Multi-agent reinforcement learning for thermally-restricted performance optimization on manycores," in *Design, Automation & Test in Europe Conference & Exhibition, DATE*. IEEE, 2024, pp. 1–6.

[11] D. Huang, L. Costero, and D. Atienza, "Is the powersave governor really saving power?" in *24th International Symposium on Cluster, Cloud and Internet Computing, CCGrid*. IEEE, 2024, pp. 273–283.

[12] S. Kim, K. Bin, S. Ha, K. Lee, and S. Chong, "ztt: learning-based DVFS with zero thermal throttling for mobile devices," in *19th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys*. ACM, 2021, pp. 41–53.

[13] A. Zou, Y. Ma, K. Garimella, B. Lee, C. D. Gill, and X. Zhang, "F-LEMMA: fast learning-based energy management for multi-/many-core processors," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 42, no. 2, pp. 616–629, 2023.

[14] Q. Zhou, Y. Zhang, X. Xu, Q. Zhang, H. Wu, Y. Lian, and Y. Zhao, "A lightweight drdpg-based RL DVFS for video rendering on CPU-GPU integrated soc," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 71, no. 5, pp. 2119–2131, 2024.

[15] J. Li, W. Jiang, Y. He, Q. Yang, A. Gao, Y. Ha, E. Ozcan, R. Bai, T. Cui, and H. Yu, " FiDRL: Flexible Invocation-Based Deep Reinforcement Learning for DVFS Scheduling in Embedded Systems ," *IEEE Transactions on Computers*, no. 01, pp. 1–14, Sep. 5555.

[16] Y. Bai, V. W. Lee, and E. Ipek, "Voltage regulator efficiency aware power management," in *22nd International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS*. ACM, 2017, pp. 825–838.

[17] B. Donyanavard, T. Mück, A. M. Rahmani, N. D. Dutt, A. Sadighi, F. Maurer, and A. Herkersdorf, "SOSA: self-optimizing learning with self-adaptive control for hierarchical system-on-chip management," in *52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. ACM, 2019, pp. 685–698.

[18] R. Jain, P. R. Panda, and S. Subramoney, "Cooperative multi-agent reinforcement learning-based co-optimization of cores, caches, and on-chip network," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 4, pp. 32:1–32:25, 2017.

[19] L. Costero, A. Iranfar, M. Zapater, F. D. Igual, K. Olcoz, and D. Atienza, "Resource management for power-constrained HEVC transcoding using reinforcement learning," *IEEE Trans. Parallel Distributed Syst.*, vol. 31, no. 12, pp. 2834–2850, 2020.

[20] G. Pan, J. Jou, and B. Lai, "Scalable power management using multilevel reinforcement learning for multiprocessors," *ACM Trans. Design Autom. Electr. Syst.*, vol. 19, no. 4, pp. 33:1–33:23, 2014.

[21] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *32nd Conference on Artificial Intelligence, AAAI*. AAAI Press, 2018, pp. 2974–2982.

[22] A. R. Alameldeen and D. A. Wood, "IPC considered harmful for multiprocessor workloads," *IEEE Micro*, vol. 26, no. 4, pp. 8–17, 2006.

[23] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, 1992.

[24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *33nd International Conference on Machine Learning, ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, 2016, pp. 1928–1937.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR*, 2016.

[26] M. Andrychowicz, A. Raichuk, P. Stanczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters in on-policy reinforcement learning? A large-scale empirical study," *CoRR*, vol. abs/2006.05990, 2020.

[27] "Intel® core™ i9-12900kf processor," https://www.intel.com/content/www/us/en/products/sku/134600/intel-core-i912900kf-processor-30m-cache-up-to-5-20-ghz/specifications.html, Intel Corporation, 2021, (accessed: 2024-11-15).

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *32th Annual Conference on Neural Information Processing Systems, NeurIPS*, 2019, pp. 8024–8035.

[29] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *17th International Conference on Parallel Architectures and Compilation Techniques, PACT*. ACM, 2008, pp. 72–81.

[30] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *22nd International Symposium on Computer Architecture, ISCA*. ACM, 1995, pp. 24–36.

[31] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *35th International Conference on Machine Learning, ICML*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 1582–1591.

333-eDRAM – 3T Embedded DRAM Leveraging Monolithic 3D Integration of 3 Transistor Types: IGZO, Carbon Nanotube and Silicon FETs

David Kong[1†], Shvetank Prakash[1], Jedrzej Kufel[2], Georgios Kyriazidis[1], Yasmine Omri[1], David Verity[2], Emre Ozer[2], Vijay Janapa Reddi[1], and Gage Hills[1]
[1]Harvard University, MA, USA; [2]Pragmatic Semiconductor, Cambridge, UK; [†]Email: dkong@g.harvard.edu

***Abstract*—The memory wall is a major bottleneck for continuing to improve the energy efficiency of computing systems. To overcome this challenge, various nanomaterials, devices, circuits, architectures, and three-dimensional (3D) integration techniques are under development for future memory solutions. However, major trade-offs exist when designing memories to achieve high on-chip memory capacity, high retention time, high endurance, low access times, low access energy, and low static leakage power. We present an energy- and area-efficient embedded DRAM memory architecture (quantified by EADP: the product of total energy consumption, circuit area footprint and application execution time) that leverages monolithic three-dimensional (3D) integration of three types of field-effect transistors (FETs): (i) Indium Gallium Zinc Oxide (IGZO) FETs for ultra-low off-state leakage currents enabling high retention time DRAM; (ii) Carbon Nanotube FETs (CNFETs) for high on-state drive currents leading to fast access times; and (iii) Silicon CMOS for its combined energy efficiency and low off-state leakage current (for memory peripheral circuits implemented on the bottom physical circuit layer). Our resulting 333-eDRAM achieves each of the following simultaneously, which we quantify and describe how to co-optimize in this paper: *high density*, *high retention time*, *high endurance*, *low access times*, *low access energy*, and *low static leakage power*. We show full physical layout designs detailing how to implement 333-eDRAM and quantify EADP for an ARM Cortex-M0 processor + on-chip 333-eDRAM implemented at a 7 nm technology node, running applications from the Embench benchmark suite. Using cycle-accurate simulations of applications, SPICE circuit simulations, compact models calibrated to experimental data, and detailed full physical layout designs of 333-eDRAM memories, we show that on average (across 16 Embench benchmarks), ARM Cortex-M0 + IGZO/CNT/Si 333-eDRAM offers 1.96× better EDP and 5.15× better EADP than ARM Cortex-M0 + Silicon eDRAM.**

I. Introduction

CONVENTIONAL transistor scaling is giving diminishing returns for improving energy efficiency of computation and memory. Computing performance has become limited by *system-level bottlenecks*; in particular, the *memory wall*, in which a wide range of computing systems can spend the vast majority (>90%) of their execution time and energy simply passing data back-and-forth to off-chip memory without doing useful computation [1]. To overcome the memory wall, a wide range of technologies to integrate massive memory *on-chip* are being explored, targeting ultra-dense and fine-grained connectivity between computation and memory. Technologies under development include: beyond-CMOS devices (e.g., various non-volatile memories), advanced integration techniques (e.g., integrating chiplets on interposers, various three-dimensional (3D) integration and 3D packaging techniques [2]–[4]), and alternative semiconductor nanomaterials (e.g., carbon nanotubes (CNTs), two-dimensional materials (MoS_2, WSe_2, hBN), and thin-film oxides (IGZO, SnO) [5]–[8]).

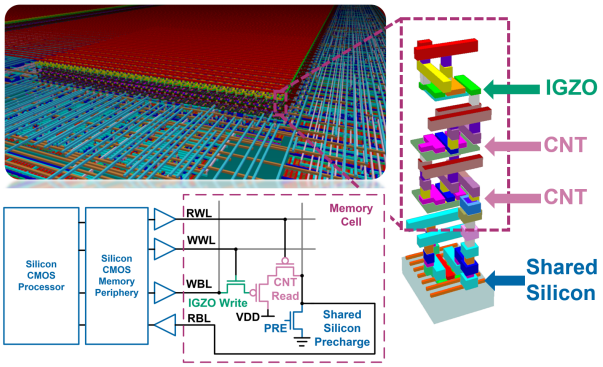


Fig. 1. 333-eDRAM from schematic to bitcell layout to rendering of full physical implementation of ARM Cortex-M0 + 333-eDRAM.

Among these technology options, embedded (on-chip) Dynamic RAM (eDRAM) using Indium Gallium Zinc Oxide (IGZO) field-effect transistors (FETs) has recently emerged as a promising memory solution for a wide range of low power and high performance applications. IGZO is an attractive material for eDRAM since:

- Its wide bandgap (≈ 3.5 eV) enables IGZO FETs (NMOS) to exhibit ultra-low leakage current (when the gate-to-source voltage, V_{GS}, is significantly below the threshold voltage, V_T). This enables ultra-high retention times for IGZO eDRAM cells (e.g., >1,000 seconds has been shown experimentally [9]), even without large capacitors to hold the state of each storage node (i.e., no deep trench capacitors or pillar capacitors), which reduces read/write energy.
- IGZO FETs can be fabricated at low processing temperatures (< 300 °C [8]), enabling monolithic 3D integrated circuits (ICs), in which multiple layers of IGZO FETs can be fabricated sequentially over the same starting substrate. Monolithic 3D integration permits ultra-dense and fine-grained connectivity between vertical circuit layers, since these layers are connected using standard high-density metal vias from the back-end-of-line (BEOL). Notably, IGZO can be fabricated on top of industry-standard silicon

CMOS (e.g., eDRAM on top of a processor core in silicon CMOS [10]), together with other FETs that also can be fabricated at low temperatures in the BEOL (such as carbon nanotube FETs or CNFETs), for high memory bandwidth and density (bits per footprint area).

Despite IGZO's benefits, today's IGZO FETs suffer from relatively low mobility ($<$ 100 cm^2/V.s [11]–[13]) vs. silicon FETs. This results in low on-state drive currents of IGZO FETs, leading to slow access times, and ultimately degrades energy efficiency at the application level. To overcome the challenges of IGZO eDRAM — and realize an energy- and area-efficient memory (details below) – we present ***333-eDRAM***, a 3-Transistor eDRAM circuit topology that leverages Monolithic 3D integration of 3 types of FET technologies: IGZO FETs, CNFETs, and Silicon CMOS FETs.

- Our 333-eDRAM (Figure 1) combines the benefits of each type of FET, while circumventing their inherent challenges (details in Table I and Section II). To complement IGZO FETs, CNFETs exhibit high on-state drive currents that we leverage to improve eDRAM read latency. CNFETs can also be fabricated in the BEOL of monolithic 3D ICs.
- To leverage the benefits of IGZO FETs and CNFETs simultaneously, we implement a 3-transistor (3T) memory cell topology that comprises one IGZO FET with ultra-low leakage current leading to high retention times, and two CNFETs for fast read latency (write latency is limited by Si CMOS drive currents and is also fast).
- Multiple layers of IGZO/CNFET eDRAM cells are monolithically integrated on top of memory peripheral circuits implemented in silicon CMOS (for its combined energy efficiency and low off-state leakage current). Thus, the memory density (bit cells per footprint area) increases with the number of CNFET and IGZO FET tiers.

As we describe in this paper, our IGZO/CNT/Si 333-eDRAM enables strictly better Energy/Area/Delay (EAD) trade-offs (i.e., a strictly better Energy-Area-Delay Product (EADP) Pareto-optimal trade-off curve) vs. silicon eDRAM for embedded computing systems. Here, E = total chip energy consumption, A = total circuit footprint area, and D = total application runtime. Our key contributions are:

1) **333-eDRAM circuit topology that leverages the benefits of three FET types (IGZO FETs, CNFETs, Si CMOS FETs) while circumventing their inherent challenges.** Our 333-eDRAM achieves each of the following simultaneously: (i) *high density* (due to monolithic 3D integration); (ii) *high retention time* (due to ultra-low leakage of IGZO FETs); (iii) *high endurance* (due to charge-based storage); (iv) *fast read/write access times* (due to high CNFET drive current); (v) *low energy per read/write access* (high retention time is achieved even with small capacitances of DRAM storage nodes); and (vi) *low static power consumption* (limited by DRAM peripheral circuitry).
2) **A systematic methodology to select device- & circuit-level parameters for 333-eDRAM**: threshold voltage (V_T), supply voltage (V_{DD}), overdrive voltage (V_{OV}, used during eDRAM writes), off-state leakage current (I_{OFF}), on-state drive current (I_{ON}), and data retention time (DRT) **to optimize overall system EADP**. Using our methodology, we quantify EADP for an ARM Cortex-M0 processor + on-chip 333-eDRAM implemented at a 7 nm technology node (derived from the ASAP7 process design kit [14]), running applications from the Embench suite [15]. We show that on average (across 16 Embench applications), Cortex-M0 + 333-eDRAM improves system-level EADP by 5.15$\times$ vs. Cortex-M0 + Si eDRAM. Our results leverage cycle-accurate Register Transfer Level (RTL) simulations, SPICE simulations, compact models calibrated to experimental data (for IGZO FETs, CNFETs, and Si CMOS), and wire parasitics modeled from physical layouts of our 333-eDRAM.
3) **Hybrid 333-eDRAM + SRAM** – which partitions a small fraction of the address space into silicon SRAM and the remainder into 333-eDRAM – offers additional opportunities to improve EADP. Such opportunities stem from application-specific memory access patterns: typical applications exhibit highly non-uniform distributions of required DRTs among memory addresses. For the 'PetriNet' Embench application, for example, fewer than 10% of the memory addresses require >1,000,000$\times$ higher DRT vs. the remaining 90% of the memory addresses (see Sec. IV-B). Thus, the memory addresses with the highest DRTs can be partitioned into a small silicon SRAM on the bottom circuit layer, while the 333-eDRAM cells in the BEOL can be optimized for a less stringent DRT (instead of the worst case DRT), improving overall EADP.

II. 333-eDRAM Memory Architecture

Figure 2 shows the 333-eDRAM schematic. The bottom circuit layer comprises the eDRAM peripheral circuits (address decoder, precharge circuitry, sense amplifiers), which are implemented in Si CMOS. Multiple tiers of CNFETs and IGZO FETs are monolithically integrated on top of the Si CMOS. The 3T cell topology is designed to leverage the benefits of CNFETs and IGZO FETs for the separate read and write paths (details below). Each 3T cell comprises 1 IGZO FET and 2 CNFETs spanning three circuit tiers. The physical layout is designed so that multiple layers of IGZO/CNT eDRAM cells can be fabricated directly on top of each other while sharing the same Si CMOS peripherals on the bottom.

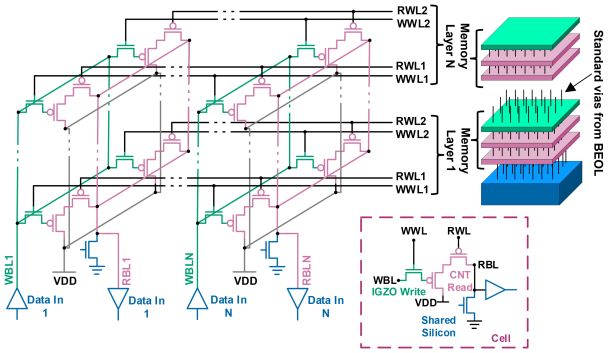


Fig. 2. Circuit schematic for our IGZO/CNT/Si 333-eDRAM comprising IGZO FETs (green), CNFETs (pink), and Si CMOS (blue).

Table I summarizes key benefits and challenges for IGZO FETs, CNFETs, and Si CMOS FETs. 333-eDRAM is designed to leverage each benefit, and circumvent each challenge.

TABLE I
FET BENEFITS (+) AND CHALLENGES (–).

CNFET	(+) high drive current (+) BEOL-compatible (low temperature fabrication) (–) susceptible to metallic CNTs [5] (increases I_{OFF})
IGZO FET	(+) ultra-low leakage current (+) BEOL-compatible (low temperature fabrication) (–) low drive current (low mobility)
Si CMOS	(+) high drive current (+) low leakage current (–) bottom layer only (high temperature fabrication)

To develop intuition for the benefits of 333-eDRAM, we refer to the following circuit parameters (as shown in Fig. 3): C_{RWL}, C_{WWL}, C_{RBL}, and C_{WBL} are the lumped capacitances of the Read Word Line (RWL), Write Word Line (WWL), Read Bit Line (RBL), and Write Bit Line (WBL), respectively (including both wire capacitance, and capacitances associated with the access transistors connected to the wordlines/bitlines); R_{RBL} and R_{WBL} are the lumped wires resistances of the bitlines; C_{SN} is the lumped capacitance of the Storage Node (SN); $R_{EFF,IGZO}$, $R_{EFF,CNFET}$, and $R_{EFF,Si}$ are the effective drive resistances of each FET; and $I_{OFF,IGZO}$, $I_{OFF,CNFET}$, and $I_{OFF,Si}$ are the off-state leakage currents of each FET. We first use these effective C, R, and I terms simply to provide insight, and then describe how we leverage industry-standard Electronic Design Automation (EDA) tools (circuits simulators + compact models) to precisely quantify each of the metrics listed below:

- *Fast read latency*: the charge on SN determines whether the CNFET is "on" or "off". The read latency is limited by the time it takes to charge the RBL (Figure 3a), and is proportional to $R_{EFF,CNFET}C_{RBL}$, where $R_{EFF,CNFET}$ is small (vs. $R_{EFF,IGZO}$), resulting in fast read latency.
- *Fast write latency*: the time to charge the SN is affected by $R_{EFF,Si}$, R_{WBL}, and $R_{EFF,IGZO}$. Despite the high $R_{EFF,IGZO}$ being in the write path, its contribution to overall write delay is proportional to $R_{EFF,IGZO}C_{SN}$ (evident using an Elmore delay model [16]), which is relatively small since C_{SN} is small. Thus, write delay is primarily limited by the relatively small $R_{EFF,Si}$ (Figure 3b).
- *Low read and write energy*: dynamic read energy is dominated by C_{RBL} and C_{RWL} (and write energy by C_{WBL} and C_{WWL}), and thus is not significantly impacted by the choice of transistors in the read/write paths.
- *Low static power consumption*: for eDRAM, static power consumption is limited by leakage through the peripheral circuitry, since there is no direct path from V_{DD} to ground inside each cell (unlike SRAM). The peripheral circuits are implemented using Si CMOS so leakage power is limited by $I_{OFF,Si}$ (where high-V_T can be utilized).
- *High Retention Time*: proportional to $C_{SN}/I_{OFF,IGZO}$, which is high due to ultra-low $I_{OFF,IGZO}$.

To *quantify* each of the above metrics, we create a custom Process Design Kit (PDK) for monolithic 3D ICs comprising a reconfigurable number of CNFET tiers, IGZO FET tiers, and metal routing layers on top of Si CMOS at a 7 nm node. Our PDK is derived from the ASAP7 PDK [14]. To add additional CNFET and IGZO FET tiers, we insert additional layers into the ASAP7 layer stack, many of which are copies of existing layers (e.g., 'M1' and 'M2' are replicated above upper-layer CNFETs and IGZO FETs). Our PDK enables industry-standard EDA tools to extract CNFETs and IGZO FETs from our custom layouts into circuit netlists, which we simulate using Cadence Spectre. For circuit simulations using each FET: we leverage the ASAP7 models for Si CMOS, and we leverage the Virtual Source CNFET model from references [17] and [18] for CNFETs. For IGZO FETs, we also leverage a virtual source FET model [19], which facilitates adjusting key FET design parameters (e.g., V_T, mobility, subthreshold slope), which we calibrate using experimental data. Figure 4 illustrates example drain current vs. gate-to-source voltage (I_D vs. V_{GS}) relationships from these compact models.

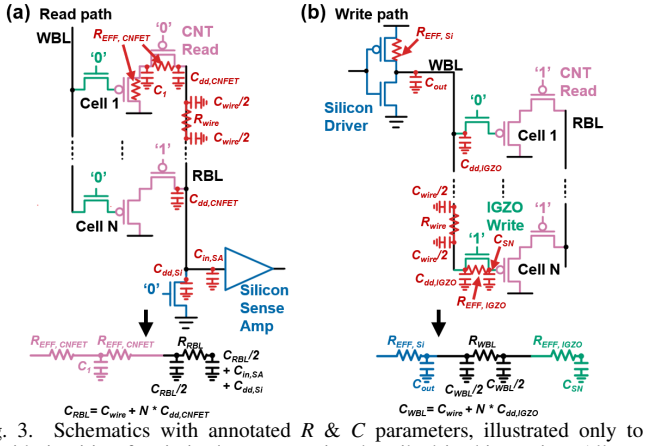


Fig. 3. Schematics with annotated R & C parameters, illustrated only to provide intuition for timing/energy metrics described in this section. All our quantitative results are extracted from detailed circuit simulations, instead of these simple models (see Sec. III). $C_{dd,IGZO}$, $C_{dd,CNFET}$, and $C_{dd,Si}$ indicate the effective capacitance seen on the drain terminal of IGZO FETs, CNFETs, and Si FETs. $C_{in,SA}$ is the input capacitance of the Sense Amp. C_1 is the effective capacitance on the node it annotates. C_{out} is the effective output capacitance of the Si driver for the WBL.

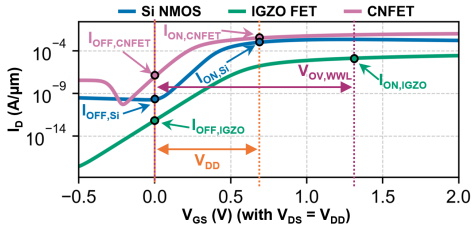


Fig. 4. Example I_D vs. V_{GS} from compact models in this work. I_{OFF} is measured at $V_{GS} = 0$, I_{ON} is measured at $V_{GS} = V_{DD}$ for CNFET/Si, and at $V_{GS} = V_{OV,WWL}$ (wordline overdrive voltage) for IGZO.

The IGZO FET model in Figure 4 is intended to be representative of typical experimentally measured results (see Table II). We choose *conservative* values – as opposed to best-case values – for FET parameters. For example, although subthreshold slope (SS) of 74.4 mV/decade and mobility of 34 $cm^2/V.s$ have been experimentally measured from IGZO FETs with 38 nm gate length [20], our conservative IGZO FET models exhibit SS of 90 mV/decade (worse) and mobility of 10 $cm^2/V.s$ (worse) for 44 nm (larger) gate length.

TABLE II
TYPICAL VALUES FOR I_{ON} AND I_{OFF} FROM EXPERIMENTALLY MEASURED IGZO FETS, CNFETS, AND SI CMOS (ΔV_{GS} IS THE GATE-TO-SOURCE VOLTAGE USED TO MEASURE I_{OFF} AND I_{ON}).

Type	$I_{ON}(A/\mu m)$	$I_{OFF}(A/\mu m)$	$\Delta V_{GS}(V)$
IGZO [11], [21]–[23] [9], [24]	$3 \cdot 10^{-5}$ to $2 \cdot 10^{-7}$	$5 \cdot 10^{-18}$ to $3 \cdot 10^{-21}$	~ 3
CNFET [5], [25]–[27]	$9.8 \cdot 10^{-4}$ to $3 \cdot 10^{-6}$	$1 \cdot 10^{-8}$ to $1 \cdot 10^{-9}$	0.5 to 1.8
7 nm FinFET Silicon [14]	$1.87 \cdot 10^{-3}$ to $4.66 \cdot 10^{-4}$	$9.70 \cdot 10^{-8}$ to $1.96 \cdot 10^{-10}$	0.7

While our models serve the purpose of providing quantifiable results in this paper, we note that designers can use alternative device models (e.g., for other technology nodes) to quantify the benefits of 333-eDRAM for their own applications. We also note that alternative FET technologies – with similar trade-offs as in Table I – can be used in the BEOL to implement 333-eDRAM. For example, alternative low-leakage FETs instead of IGZO FETs, and alternative high-performance FETs instead of CNFETs, can be used to implement 333-eDRAM following similar design guidelines.

III. System-Technology Co-Optimization for Specific Applications

To quantify the *system-level benefits* of 333-eDRAM, we compare the area, energy, and execution time of an embedded system (ARM Cortex-M0 + embedded DRAM) implemented in *two different physical technologies* at a 7 nm node:

1) **Si CMOS only**, with both the Cortex-M0 and eDRAM in Si CMOS. Silicon eDRAM is often used as an SRAM replacement for large on-chip memory (since it can have lower leakage and higher density vs. SRAM [28], [29]). However, DRAM memory cells have finite DRT which require *refresh* (periodically reading and then rewriting the values stored in memory cells). Refresh incurs significant energy overheads, which we account for in our analysis. This is the baseline design for EADP comparisons.
2) **Monolithic 3D Integration of IGZO + CNT + Si CMOS**, with the Cortex-M0 in Si CMOS, and the 333-eDRAM as described in this paper.

A. Quantifying Energy, Delay, and Area

Since applications have varying memory access patterns with different DRT requirements, it is insufficient to simply consider read/write energy/latency of the memory alone to characterize a system. The large number of design parameters at the device, circuit, and application levels leads to a large number of potential designs (one for each combination of parameter values) for 333-eDRAM. Figure 6 shows our design methodology. We consider the following parameters:

- **FET Parameters** – for 333-eDRAM and Si-eDRAM, we sweep: (i) the threshold voltage (V_T) of each FET in the 3T bitcell; (ii) the overdrive voltage (V_{OV}) for the write wordline (WWL in Fig. 2); (iii) the worst case storage node voltage (V_{SN}, to ensure the sense amplifier can distinguish between reading "0" vs. "1"). These parameters affect trade-offs in speed, energy, and DRT. For the Cortex-M0, we optimize V_T of the FETs in digital logic gates (ASAP7 offers 4 options for V_T: SLVT, LVT, RVT, and HVT, for "Super Low," "Low," "Regular," and "High" V_T). We use V_{DD} = 0.7 V (the default for ASAP7), and Contacted Gate Pitch (CGP) = 54 nm for CNFETs (the same as Si FETs in ASAP7). The CGP of the IGZO FET does not limit the size of our 333-eDRAM bit cell, so we extend the gate length to 44 nm to improve sub-threshold slope (see Sec. II). We perform SPICE simulations to accurately quantify DRT (accounting for capacitive coupling effects and leakage paths). We export eDRAM layouts in library exchange format (.lef), so they can be instantiated with the Cortex-M0 in the physical layout of the full system.
- **Circuit Parameters** – we design, layout, and simulate 333-eDRAM cells, peripherals, subarrays, and memory controllers using commercial EDA tools (Cadence Virtuoso, Cadence Spectre, Siemens Calibre). We extract energy, delay, and area considering wire congestion and wire parasitics from physical layouts of 333-eDRAM (see Sec. III-C). We perform circuit simulations using the 333-eDRAM schematics to quantify: (i) *read/write energy* – we combine this with the total number of memory accesses, see Application Parameters, to quantify total energy consumed by the eDRAM; (ii) *read/write access time* – we design our system so that reads and writes from the eDRAMs can be completed in a single clock cycle (we partition the memories into 2 kB sub-arrays, each with 512 32-bit words, which improves timing due to smaller bitline capacitive loading to ensure that the critical path delay is shorter than the clock period of the Cortex-M0). We design the Cortex-M0 using standard Very-Large-Scale Integration (VLSI) digital design flows, including logic synthesis (Cadence Genus) and place-and-route (Cadence Virtuoso). We sweep the range of target clock frequencies (for synthesis & place-and-route tools to meet timing) from 100 MHz to 2 GHz), for each of the four V_T options. We leverage Synopsys VCS to analyze energy for application-specific traces (details below).
- **Application Parameters** – we analyze programs from Embench [15]. We compile each application in Embench [15], which we then analyze using cycle-accurate Cortex-M0 Register Transfer Level (RTL) simulations in Synopsys VCS. This allows us to: (i) extract the number of reads and writes on the memory bus to determine DRT requirements; and (ii) determine program and data memory size requirements. Figure 5 shows summary statistics for the distribution of required DRTs for the Embench applications.

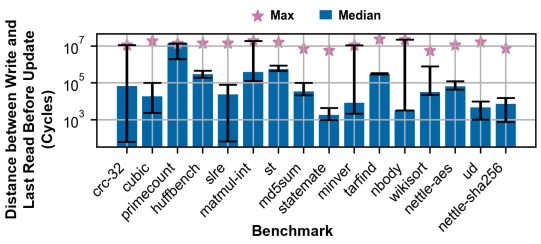


Fig. 5. Median required DRT across memory addresses, shown for the Embench applications running on Cortex-M0. The upper and lower error bars show the DRT required by 75% and 25% of memory addresses, respectively.

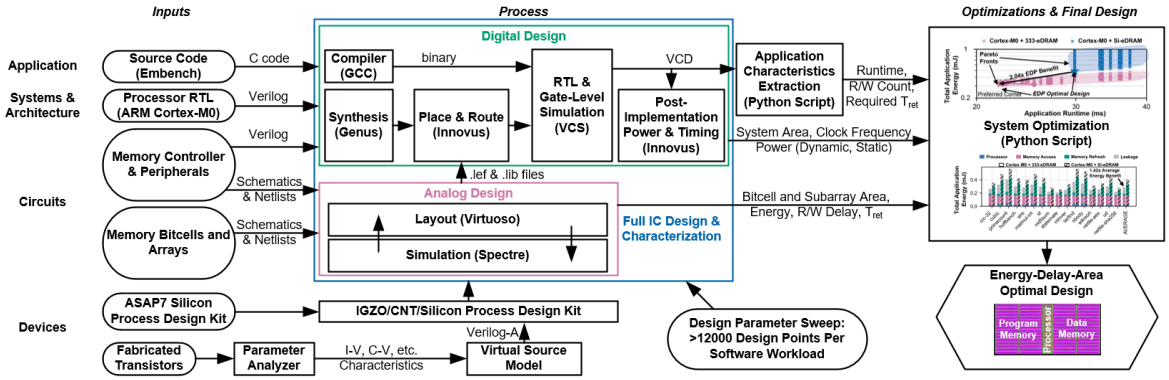


Fig. 6. Design methodology for 333-eDRAM and system-level energy efficiency optimization.

B. Impact of Process Variations on eDRAM

Instead of designing for the nominal case, i.e., with no variations in any of the FET parameters, eDRAM is sensitive to process variations, which must be considered during design time. For example, variations in IGZO FET V_T lead to exponential variations in DRT, since DRT is limited by the sub-threshold leakage of the IGZO FET. Figure 7 illustrates the impact of V_T variations on DRT, showing the degradation of charge on the storage node as a function of time for 1,024 Monte Carlo (MC) simulations. Without considering variations, a naïve nominal design for a 7 nm node 333-eDRAM might target 1.72 ms retention time (for example), by selecting C_{SN} = 160 aF and V_T such that $I_{OFF,IGZO}$ = 48.9 fA (for V_{DD} = 0.7 V , gate length: L_G = 44 nm, and FET width: W = 72 nm). When considering variations in V_T, e.g., with standard deviation σ_{V_T} = 15 mV centered around the nominal V_T (the target V_T with no variations), the distribution of DRT among 3T memory cells varies by over an order of magnitude: from 0.496 ms to 5.75 ms for the MC analysis in Figure 7. That is $> 3.3\times$ above the nominal design target of DRT = 1.72 ms, and $> 3.4\times$ below. This emphasizes the importance of considering variations while selecting target device- and circuit-level parameters for 333-eDRAM. In particular, constraints on the distribution of DRTs should be targeted by designers, e.g., target a specific value of V_T such that 99.9999% of 3T memory cells meet a desired DRT. For our results in Sec. IV, we analyze V_T within ± 50 mV of its nominal value. Note that, redundancy and error correction techniques can also be used with 333-eDRAM.

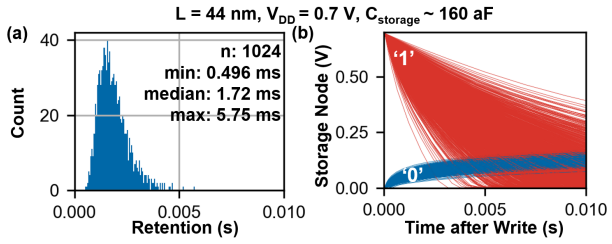


Fig. 7. MC simulation results of IGZO-CNT-Si 333-eDRAM (1,024 MC samples, corresponding to a 1 kbit array). (a) DRT histogram for a data-integrity threshold of 0.5 V (with $V_{DD} = 0.7\ V$). (b) Storage Node voltage during data hold. DRT can be further improved by adjusting V_{DD} and the target V_T to meet DRT distribution requirements.

C. 333-eDRAM Physical Implementation

Figure 8 illustrates the eDRAM bitcell layouts (Si CMOS & 333-eDRAM), including the vertical connections between different CNFET and IGZO FET tiers in three dimensions. Figure 9 shows the layout of a 2 kB sub-array (333-eDRAM), which we instantiate multiple times to create the 64 kB memories (Fig. 10) to run the Embench applications (program memory & data memory). Physical dimensions are in Table III.

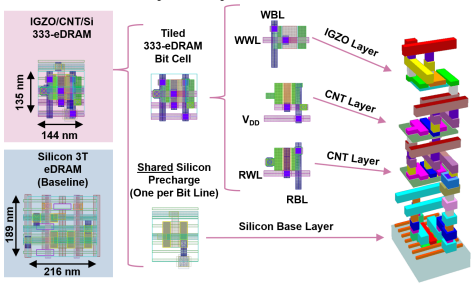


Fig. 8. Physical layouts of 333-eDRAM vs. Si eDRAM bitcells. The blue box indicates the bitcell boundary that can be tiled across the array.

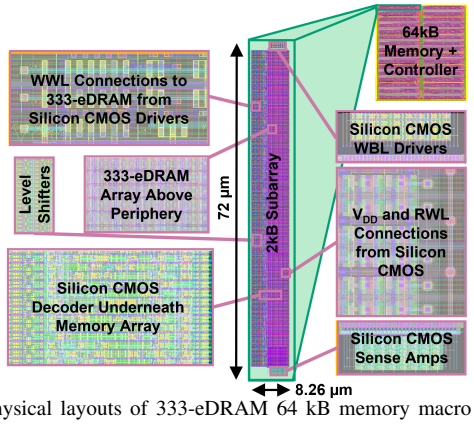


Fig. 9. Physical layouts of 333-eDRAM 64 kB memory macro comprising 2 kB subarrays (512 words of 32-bits). Memory peripherals (wordline drivers including level shifters to overdrive the WWL, pre-charge circuits, decoders) occupy the Si CMOS directly underneath the CNFET/IGZO FET bitcells.

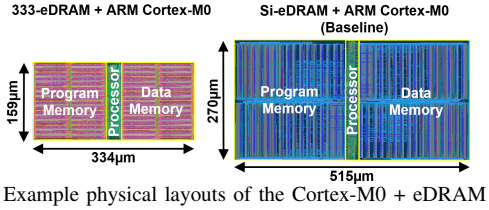


Fig. 10. Example physical layouts of the Cortex-M0 + eDRAM systems.

IV. Results

A. EADP Benefits for Embedded Systems

Figure 11 illustrates trade-offs in application execution time vs. total energy consumption (considering application-specific RTL simulations), as a result of sweeping the FET-level, circuit-level, and application-level parameters (Sec. III). Each point in Fig. 11 corresponds to a unique combination of V_T (for FETs in e-DRAM cells), V_T (for FETs in digital logic), V_{OV}, V_{SN}, and target clock frequency during synthesis/place-and-route. The design with the best EDP is selected for each system (Si CMOS vs. IGZO/CNT/Si) to compare energy efficiency. The IGZO/CNT/Si implementation improves EDP by 2.04× (for the 'matmul-int' application). Figure 12 shows how energy is attributed to different sub-systems (for the EDP-optimal design for each application).

Finally, Figure 13 illustrates the overall EADP benefit of the IGZO/CNT/Si system for each Embench application. Considering both the energy efficiency (EDP) benefit, as well as the area benefit of 333-eDRAM, the IGZO/CNT/Si exhibits 5.15× better EADP on average (across all Embench applications), for the systems we analyze in this paper. On average, the power densities of the systems are 10.36 W/cm^2 and 26.19 W/cm^2 for the Si-only system and the IGZO/CNT/Si system, respectively, which fall within typical thermal budget constraints [1]. Additional details about the EADP optimal designs are Table III.

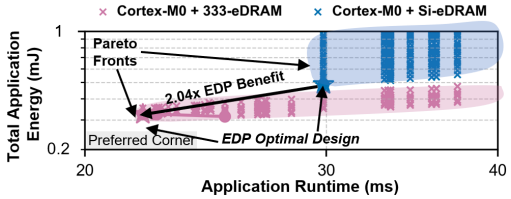


Fig. 11. Total energy vs. execution time for the 'matmul-int' application. >12,000 combinations of parameters are analyzed. We only show the designs with eDRAM access time shorter than the Cortex-M0 critical path delay (ensuring single-cycle memory accesses).

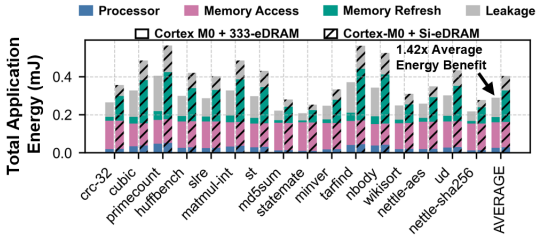


Fig. 12. Energy breakdown of EDP optimal designs for each application. On average, Cortex-M0 + 333-eDRAM improves energy by 1.42×.

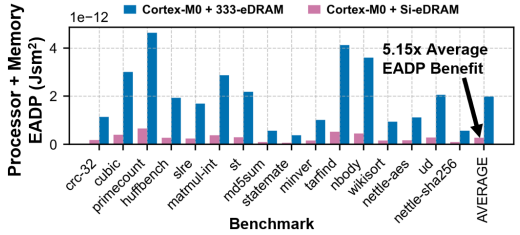


Fig. 13. On average, Cortex-M0 + 333-eDRAM improves EADP by 5.15×.

TABLE III
EADP Optimal Design Summary for the 'matmul-int' Workload

System	Cortex-M0 + Si eDRAM	Cortex-M0 + 333-eDRAM
'matmul-int' Runtime (cycles)	20,047,348	20,047,348
Required Data Memory DRT (cycles)	19,594,623	19,594,623
Clock frequency	671 MHz	909 MHz
ARM Cortex-M0 V_T	SLVT	HVT
Supply Voltage, Overdrive Voltage	0.7 V, 1.0 V	0.7 V, 1.4 V
ARM Cortex-M0 Energy/Cycle	2.03 pJ	1.67 pJ
Memory Energy/Cycle	21.3 pJ	13.6 pJ
Refresh Controller Energy/Cycle	1.03 fJ	0.89 fJ
Memory Data Retention Time	51.3 μs	501 μs*
Global Controller Delay	478 ps	396 ps
Subarray Decoder Delay	171 ps	189 ps
WWL to 95% V_{SN} Delay	328 ps	251 ps
RWL to SA Delay @ V_{SN}=0.5 V	793 ps	489 ps
Bitcell Area	0.041 μm^2	0.019 μm^2
2 kB Sub-Array Area	1641.5 μm^2	594.7 μm^2
64 kB Memory Area	0.068 mm^2	0.025 mm^2
Total Area (Core + 128 kB Memory)	0.139 mm^2	0.053 mm^2

*Worst cell with ±50 mV V_T for slowest write and shortest DRT.

B. Hybrid 333-eDRAM + SRAM

Figure 14 illustrates that the distribution of required DRTs in an application can be highly non-uniform. For the 'Petri-Net' application, for example, fewer than 10% of the memory addresses require >1,000,000× higher DRT vs. the remaining 90% of the memory addresses. A *Hybrid 333-eDRAM + SRAM* approach offers additional opportunities to improve EADP, by: (i) partitioning the memory addresses with the highest required DRT into a small silicon SRAM on the bottom circuit layer; and (ii) optimizing the 333-eDRAM cells in the BEOL for a less stringent DRT requirement for the remaining memory addresses (e.g., by lowering V_T to improve access time at the cost of degraded retention time). Such partitioning can be achieved using common compilation-time memory arrangement techniques for refresh minimization [30], [31].

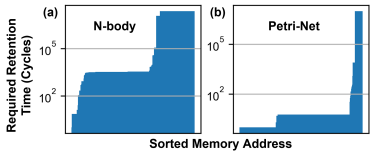


Fig. 14. The cumulative distribution function of required DRT among memory addresses (shown for two Embench applications) is highly non-uniform.

V. Conclusion

Our key contribution is the design of *333-eDRAM*: an embedded memory that combines the benefits of three FET types: IGZO FETs for ultra-low leakage & BEOL fabrication, CNFETs for high drive current & BEOL fabrication, and Si CMOS for energy-efficient memory peripherals on the bottom circuit layer. The combined result is that 333-eDRAM achieves each of the following simultaneously: high density, high retention time, high endurance, low access times, low access energy, and low static leakage power. Additionally, alternative FETs with similar trade-offs can be used to implement 333-eDRAM (e.g., 2D materials for high current FETs or low leakage FETs in the BEOL), following similar design guidelines.

Acknowledgment

We acknowledge incredibly valuable support from the U.S. National Science Foundation, Award Number 2346435.

REFERENCES

[1] M. M. S. Aly, T. F. Wu, A. Bartolo, Y. H. Malviya, W. Hwang, G. Hills, I. Markov, M. Wootters, M. M. Shulaker, H.-S. P. Wong *et al.*, "The N3XT approach to energy-efficient abundant-data computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 19–48, 2018.

[2] C. C. Sudarshan, N. Matkar, S. Vrudhula, S. S. Sapatnekar, and V. A. Chhabria, "ECO-CHIP: Estimation of Carbon Footprint of Chiplet-based Architectures for Sustainable VLSI," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 671–685.

[3] J. H. Lau, "Recent advances and trends in advanced packaging," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 12, no. 2, pp. 228–252, 2022.

[4] J. Zhao, G. Sun, G. H. Loh, and Y. Xie, "Optimizing GPU energy efficiency with 3D die-stacking graphics memory and reconfigurable memory interface," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 4, pp. 1–25, 2013.

[5] G. Hills, C. Lau, A. Wright, S. Fuller, M. D. Bishop, T. Srimani, P. Kanhaiya, R. Ho, A. Amer, Y. Stein *et al.*, "Modern microprocessor built from complementary carbon nanotube transistors," *Nature*, vol. 572, no. 7771, pp. 595–602, 2019.

[6] S.-X. Guan, T. H. Yang, C.-H. Yang, C.-J. Hong, B.-W. Liang, K. B. Simbulan, J.-H. Chen, C.-J. Su, K.-S. Li, Y.-L. Zhong *et al.*, "Monolithic 3D integration of back-end compatible 2D material FET on Si FinFET," *npj 2D Materials and Applications*, vol. 7, no. 1, p. 9, 2023.

[7] Y. Su, M. Shi, J. Tang, Y. Li, Y. Du, R. An, J. Li, Y. Li, J. Yao, R. Hu *et al.*, "Monolithic 3-D Integration of Counteractive Coupling IGZO/CNT Hybrid 2T0C DRAM and Analog RRAM-Based Computing-In-Memory," *IEEE Transactions on Electron Devices*, 2024.

[8] E. Ozer, J. Kufel, S. Prakash, A. Raisiardali, O. Kindgren, R. Wong, N. Ng, D. Jausseran, F. Alkhalil, D. Kong *et al.*, "Bendable non-silicon RISC-V microprocessor," *Nature*, pp. 1–6, 2024.

[9] A. Belmonte, S. Kundu, S. Subhechha, A. Chasin, N. Rassoul, H. Dekkers, H. Puliyalil, F. Seidel, P. Carolan, R. Delhougne *et al.*, "Lowest $I_{OFF} < 3 \times 10^{-21} A/\mu m$ in capacitorless DRAM achieved by Reactive Ion Etch of IGZO-TFT," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2023, pp. 1–2.

[10] T. Naito, T. Ishida, T. Onoduka, M. Nishigoori, T. Nakayama, Y. Ueno, Y. Ishimoto, A. Suzuki, W. Chung, R. Madurawe, S. Wu, S. Ikeda, and H. Oyamatsu, "World's first monolithic 3D-FPGA with TFT SRAM over 90nm 9 layer Cu CMOS," in *2010 Symposium on VLSI Technology*, 2010, pp. 219–220.

[11] S. Subhechha, N. Rassoul, A. Belmonte, H. Hody, H. Dekkers, M. J. van Setten, A. Chasin, S. H. Sharifi, S. Sutar, L. Magnarin *et al.*, "Ultra-low Leakage IGZO-TFTs with Raised Source/Drain for V_t > 0 V and I_{on}>30 μA/μm," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2022, pp. 292–293.

[12] H. Hosono, "How we made the IGZO transistor," *Nature Electronics*, vol. 1, no. 7, pp. 428–428, 2018.

[13] Z. Pan, Y. Hu, J. Chen, F. Wang, Y. Jeong, D. P. Pham, and J. Yi, "Approaches to Improve Mobility and Stability of IGZO TFTs: A Brief Review," *Transactions on Electrical and Electronic Materials*, pp. 1–9, 2024.

[14] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[15] "Embench™: Open Benchmarks for Embedded Platforms," 2021. [Online]. Available: https://github.com/embench/embench-iot

[16] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, no. 1, pp. 55–63, 1948.

[17] C.-S. Lee, E. Pop, A. D. Franklin, W. Haensch, and H.-S. Wong, "A compact virtual-source model for carbon nanotube fets in the sub-10-nm regime—part i: Intrinsic elements," *IEEE transactions on electron devices*, vol. 62, no. 9, pp. 3061–3069, 2015.

[18] C.-S. Lee, E. Pop, A. D. Franklin, W. Haensch, and H.-S. P. Wong, "A compact virtual-source model for carbon nanotube fets in the sub-10-nm regime—part ii: Extrinsic elements, performance assessment, and design optimization," *IEEE Transactions on Electron Devices*, vol. 62, no. 9, pp. 3070–3078, 2015.

[19] A. Khakifirooz, O. M. Nayfeh, and D. Antoniadis, "A simple semiempirical short-channel MOSFET current–voltage model continuous across all regions of operation and employing only physical parameters," *IEEE Transactions on Electron Devices*, vol. 56, no. 8, pp. 1674–1680, 2009.

[20] S. Samanta, K. Han, C. Sun, C. Wang, A. V.-Y. Thean, and X. Gong, "Amorphous IGZO TFTs featuring extremely-scaled channel thickness and 38 nm channel length: Achieving record high Gm, max of 125 μS/μm at VDS of 1 V and ION of 350 μA/μm," in *2020 IEEE Symposium on VLSI Technology*. IEEE, 2020, pp. 1–2.

[21] A. Belmonte, H. Oh, N. Rassoul, G. Donadio, J. Mitard, H. Dekkers, R. Delhougne, S. Subhechha, A. Chasin, M. Van Setten *et al.*, "Capacitor-less, long-retention (>400s) DRAM cell paving the way towards low-power and high-density monolithic 3D DRAM," in *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2020, pp. 28–2.

[22] A. Belmonte, H. Oh, S. Subhechha, N. Rassoul, H. Hody, H. Dekkers, R. Delhougne, L. Ricotti, K. Banerjee, A. Chasin *et al.*, "Tailoring IGZO-TFT architecture for capacitorless DRAM, demonstrating $> 10^3$ s retention, $> 10^{11}$ cycles endurance and L_g scalability down to 14nm," in *2021 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2021, pp. 10–6.

[23] D. Ha, W. Lee, M. Cho, M. Terai, S.-W. Yoo, H. Kim, Y. Lee, S. Uhm, M. Ryu, C. Sung *et al.*, "Highly Manufacturable, Cost-Effective, and Monolithically Stackable $4F^2$ Single-Gated IGZO Vertical Channel Transistor (VCT) for sub-10nm DRAM," in *2023 International Electron Devices Meeting (IEDM)*. IEEE, 2023, pp. 1–4.

[24] S. Subhechha, N. Rassoul, A. Belmonte, R. Delhougne, K. Banerjee, G. Donadio, H. Dekkers, M. van Setten, H. Puliyalil, M. Mao *et al.*, "First demonstration of sub-12 nm L_g gate last IGZO-TFTs with oxygen tunnel architecture for front gate devices," in *2021 Symposium on VLSI Technology*. IEEE, 2021, pp. 1–2.

[25] G. Hills, M. G. Bardon, G. Doornbos, D. Yakimets, P. Schuddinck, R. Baert, D. Jang, L. Mattii, S. M. Y. Sherazi, D. Rodopoulos *et al.*, "Understanding energy efficiency benefits of carbon nanotube field-effect transistors for digital VLSI," *IEEE Transactions on Nanotechnology*, vol. 17, no. 6, pp. 1259–1269, 2018.

[26] M. M. Shulaker, G. Pitner, G. Hills, M. Giachino, H.-S. P. Wong, and S. Mitra, "High-performance carbon nanotube field-effect transistors," in *2014 IEEE International Electron Devices Meeting*. IEEE, 2014, pp. 33–6.

[27] G. Pitner, Z. Zhang, Q. Lin, S.-K. Su, C. Gilardi, C. Kuo, H. Kashyap, T. Weiss, Z. Yu, T.-A. Chao *et al.*, "Sub-0.5 nm interfacial dielectric enables superior electrostatics: 65 mV/dec top-gated carbon nanotube FETs at 15 nm gate length," in *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2020, pp. 3–5.

[28] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology comparison for large last-level caches (L 3 Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM," in *2013 IEEE 19th international symposium on high performance computer architecture (HPCA)*. IEEE, 2013, pp. 143–154.

[29] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "RANA: Towards efficient neural acceleration with refresh-optimized embedded DRAM," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 340–352.

[30] Q. Li, Y. He, J. Li, L. Shi, Y. Chen, and C. J. Xue, "Compiler-assisted refresh minimization for volatile STT-RAM cache," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2169–2181, 2014.

[31] N. Sayed, L. Mao, R. Bishnoi, and M. B. Tahoori, "Compiler-assisted and profiling-based analysis for fast and efficient STT-MRAM on-chip cache design," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 4, pp. 1–25, 2019.

Replay4NCL: An Efficient Memory Replay-based Methodology for Neuromorphic Continual Learning in Embedded AI Systems

Mishal Fatima Minhas[1], Rachmad Vidya Wicaksana Putra[2], Falah Awwad[1], Osman Hasan[3], Muhammad Shafique[2]
[1]*Electrical and Communication Engineering Department, United Arab Emirates University (UAEU), Al Ain, UAE*
[2]*eBrain Lab, New York University (NYU) Abu Dhabi, Abu Dhabi, UAE*
[3]*School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan*
{mishal.fatima, f_awwad}@uaeu.ac.ae, {rachmad.putra, muhammad.shafique}@nyu.edu,
osman.hasan@seecs.edu.pk

***Abstract*—Neuromorphic Continual Learning (NCL) paradigm leverages Spiking Neural Networks (SNNs) to enable continual learning (CL) capabilities for AI systems to adapt to dynamically changing environments. Currently, the state-of-the-art employ a memory replay-based method to maintain the old knowledge. However, this technique relies on long timesteps and compression-decompression steps, thereby incurring significant latency and energy overheads, which are not suitable for tightly-constrained embedded AI systems (e.g., mobile agents/robotics). To address this, we propose *Replay4NCL*, a novel efficient memory replay-based methodology for enabling NCL in embedded AI systems. Specifically, Replay4NCL compresses the latent data (old knowledge), then replays them during the NCL training phase with small timesteps, to minimize the processing latency and energy consumption. To compensate the information loss from reduced spikes, we adjust the neuron threshold potential and learning rate settings. Experimental results on the class-incremental scenario with the Spiking Heidelberg Digits (SHD) dataset show that Replay4NCL can preserve old knowledge with Top-1 accuracy of 90.43% compared to 86.22% from the state-of-the-art, while effectively learning new tasks, achieving 4.88x latency speed-up, 20% latent memory saving, and 36.43% energy saving. These results highlight the potential of our Replay4NCL methodology to further advances NCL capabilities for embedded AI systems.**

***Index Terms*—Spiking Neural Networks, Neuromorphic Continual Learning, Memory Replay, Efficiency, Embedded AI**

I. Introduction

The continual learning (CL) capability shown by human has inspired the developments of CL for artificial intelligence (AI) systems, e.g., through Deep Neural Networks (DNNs). CL capability is highly desired by systems that are supposed to adapt to dynamically changing environments [1]–[3], since the knowledge acquired during the off-line training (pre-training) phase may become outdated over time. This condition can result in reduced accuracy at run-time. To address this, the knowledge update is required to ensure correct functionality of the system. However, simply performing the retraining technique from scratch is not practical, as it requires huge training time and energy consumption. Besides, it also requires a full dataset including data for the new task, which is not always available due to data restriction policy [4]. Moreover, if the retraining is not performed carefully, it may also cause the *catastrophic forgetting* (CF) issue, where accuracy for the previously trained tasks significantly decrease as the system learns a new task [5]–[10], as illustrated in Fig. 1(a).

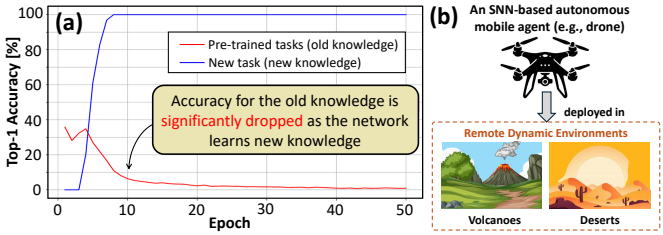


Fig. 1. (a) Accuracy of the baseline network that faces CF issues in the class-incremental learning scenario. Baseline network refers to an SNN model without any NCL capabilities. (b) Illustration of the use-case of an embedded AI system that requires energy-efficient CL capabilities, e.g., a mobile agent.

Toward this, many researches have proposed different CL methods for DNNs [2]. However, the intensive DNN operations make it challenging to efficiently enable CL capabilities, especially in the application use-cases where embedded AI systems with limited battery capacity and resources are deployed in environments that are too far or not safe for humans (e.g., mobile agents/robots like UAVs and UGVs) [11], as shown in Fig. 1(b).

In recent years, *neuromorphic continual learning* (NCL) has emerged as an attractive approach to enable efficient CL capabilities, i.e., by leveraging Spiking Neural Networks (SNNs) to perform sparse spike-based CL algorithms [12]–[17]. However, the NCL field is still very new, thereby practical solutions and methodologies for enabling efficient NCL capabilities in embedded AI systems is still not comprehensively explored. Therefore, **we mainly target the following research problem** in this paper, *how can we enable efficient NCL capabilities for resource-constrained embedded AI systems?* An efficient solution to this problem potentially enables the efficient NCL deployments for diverse applications use-cases.

A. State-of-the-Art and Their Limitations

To address CF with NCL paradigm, previous works can be categorized into two different approaches based on the learning settings, i.e., *unsupervised-* and *supervised-based learning*.

In the unsupervised learning settings, previous works have proposed different techniques, such as enhancements on bio-plausible Spike-Timing Dependent Plasticity (STDP) learning rules [13]–[16] and predictive coding [18]. However, these studies still consider static and non-event-based datasets in

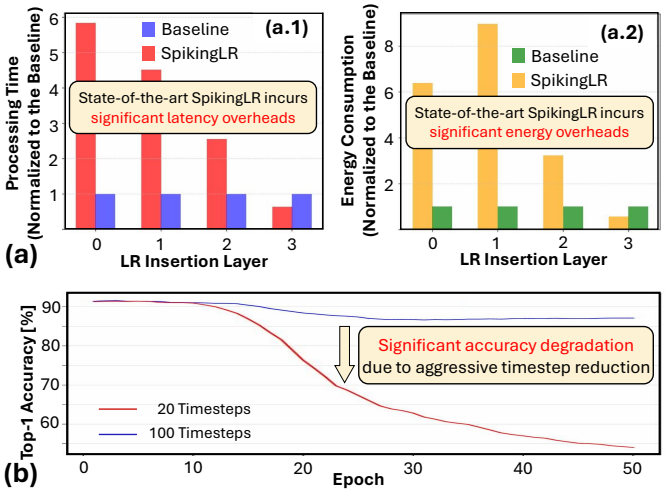


Fig. 2. (a) The state-of-the-art memory replay-based work, SpikingLR [25], incurs significant latency and energy overheads as compared to the baseline network without NCL techniques. (b) An aggressive timestep reduction (e.g., from 100 to 20 timesteps) may lead to significant accuracy degradation.

their evaluations, e.g., MNIST [19] and Fashion MNIST [20]. Therefore, their applicability for event-based neuromorphic data (e.g., from dynamic vision sensors) is still not evaluated.

Meanwhile, in the supervised learning settings, recent works have also proposed different techniques, such as active dendrites [21], architectural enhancements [22] [23], memory replay [24] [25], regularization [26], as well as Bayesian [27] and Hebbian [28] learning. Currently, the state-of-the-art work employs the memory replay-based technique to preserve high accuracy on the old knowledge (i.e., *latent knowledge*) while also learning new knowledge [25]; so-called the *spiking latent replay (SpikingLR)* for brevity. However, this technique relies on long timesteps and compression-decompression steps to achieve high accuracy [25], hence incurring significant latency and energy consumption overheads, which are not suitable for tightly-constrained embedded AI systems. These limitations indicate that *an alternate efficient memory replay-based technique is highly needed to provide NCL capabilities with low memory, latency, and energy overheads*.

To illustrate the limitations of state-of-the-art and the optimization challenges, we perform an experimental case study, which is discussed further in Section I-B.

B. Case Study and Associated Research Challenges

In this case study, we investigate the impact of a widely-used optimization technique for SNNs (i.e., timestep reduction [29]), while employing the memory replay-based technique in the class-incremental scenario. To do this, we consider the implementation of SpikingLR [25], and perform a timestep reduction accordingly (e.g., from 100 to 20 timesteps). Details of the experimental setup are provided in Section IV. The experimental results are presented in Fig. 2, from which we identify the following key observations.

- The state-of-the-art Spiking-LR work introduces huge latency and energy overheads due to its long timesteps.
- The timestep reduction may degrade the accuracy significantly, thereby hindering the implementation of optimizations on memory replay-based techniques for NCL.

Furthermore, this observation exposes several research challenges that need to be addressed, as follows.

- The processing latency should be optimized carefully so that the system can quickly respond properly to the given inputs, without a significant accuracy degradation.
- The enhancement techniques are required to compensate the loss of information due to the reduced timestep.

C. Our Novel Contributions

To address the above challenges, we propose ***Replay4NCL**, a novel methodology to enable efficient memory Replay-based NCL in resource-constrained embedded AI systems*. To achieve this, our Replay4NCL employs the following novel key design steps; see an overview in Fig. 4.

- **Perform Timestep Optimization (Section III-A):** It aims to introduce a low timestep setting into the SNN model to substantially reduce the processing latency and minimize the latent data memory.
- **Perform Parameter Adjustments (Section III-B):** It aims to compensate the loss of information due to the timestep reduction, by adjusting the parameters (e.g., neuron threshold potential and learning rate) to adapt to fewer spikes in the old (latent) data.
- **Devise a Data Insertion Strategy (Section III-C):** It aims to define the layer that will receive the latent data, so that it can effectively lead to high learning quality of latent data, and hence avoiding the CF problem.

Key Results: Our proposed methodology is evaluated using a Python implementation that runs on the Nvidia RTX 4090 Ti GPU machines, while considering the SHD dataset [30]. The experimental results on the class-incremental scenario show that our Replay4NCL maintains old knowledge with Top-1 accuracy of 90.43% compared to 86.22% from the state-of-the-art SpikingLR, while effectively learning new tasks, improving the latency by 4.88x, saving latent memory by 20%, and saving energy consumption by 36.43%.

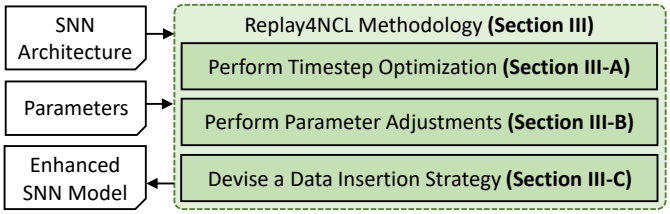


Fig. 3. An overview of our novel contributions; highlighted in green.

II. Related Work

A. Spiking Neural Networks (SNNs)

SNNs are the emerging neural network (NN) models, that utilize action potentials (i.e., spikes) to convey information and perform computation. They offer ultra-low-power/energy consumption due to their sparse spike-based operations, while achieving high performance (e.g., accuracy) [31]–[33]. SNN computations typically rely on a specific neuron model. The widely-used model is the Leaky Integrate-and-Fire (LIF) neuron [34], whose internal dynamics can be expressed as follows.

$$\tau \frac{dV_{mem}(t)}{dt} = -(V_{mem}(t) - V_{rst}) + Z(t) \quad (1)$$

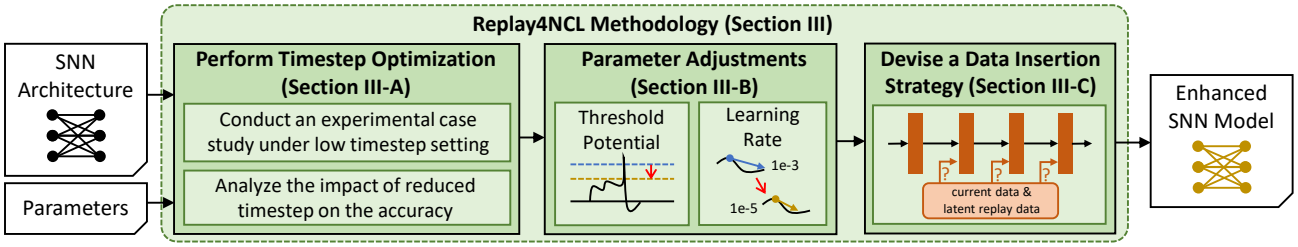


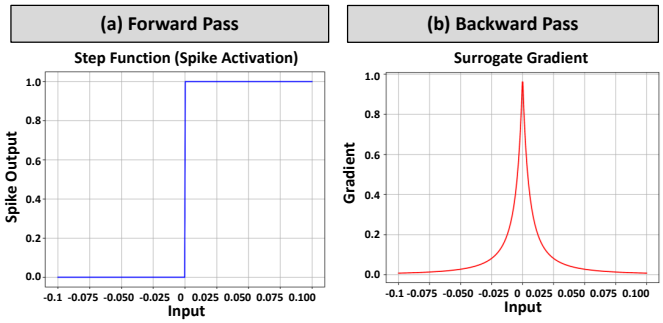
Fig. 4. Overview of our Replay4NCL methodology, showing its key steps such as timestep optimization, parameter adjustments, and data insertion strategy; the novel contributions are highlighted in green.

$$\text{if} \quad V_{mem} \geq V_{thr} \quad \text{then} \quad V_{mem} \leftarrow V_{rst} \tag{2}$$

V_{mem} is the neuron membrane potential at timestep-t, V_{rst} is the neuron reset potential, and τ is to the time constant of V_{mem} decay. Meanwhile, Z is the input, and V_{thr} is the neuron threshold potential. SNNs have advantages for their ultra-low power/energy consumption and temporal information processing by leveraging event-based data, and hence the SNN deployments on tightly-constrained embedded AI systems is highly desired.

B. Training SNNs with Surrogate Gradient (SG) Learning

The state-of-the-art SNN training under supervised settings typically involve the SG-based technique [35] [36] due to its efficacy in minimizing the loss function, e.g., using the Back-Propagation Through Time (BPTT) [37]. Therefore, in this work, we also train SNNs using the SG-based BPTT by using a fast-sigmoid surrogate gradient to approximate the step function, following the studies in [25]; see Fig. 5.



$$S(t) = \begin{cases} 1 \ if \ x(t) > \theta_{mem} \\ 0 \ if \ otherwise \end{cases} \qquad \frac{\partial S}{\partial x} \approx \frac{1}{(scale \cdot |x| + 1)^2}$$

Fig. 5. Overview of (a) the forward pass, and (b) the backward pass with SG learning technique under supervised settings.

C. Memory Replay-based Techniques for NCL

The idea of these techniques is the generation of synthetic data samples of previously learned tasks, which will be re-introduced during the NCL training phase for updating the knowledge of the system [2] [12]. This approach presents new and old tasks to the system, thereby enforcing the system to re-learn old tasks (old knowledge) through the *latent replay (LR) data* to avoid CF issues, while also learning new tasks (new knowledge). To enable NCL with memory replay, we consider employing the SNN architecture from the SpikingLR work [25], which is illustrated in Fig. 6. The network comprises of *frozen* and *learning layers*, based on where the LR data is inserted into the network. For instance, if we have L layers in the network and the LR data is inserted in layer L-1, then layer 1 until layer L-2 are the frozen layers, while layer L-1 and L are the learning layers. During the training on new data, only the learning layers are updated, while the frozen layers remain unchanged. These frozen layers simply forward the input spikes by processing them based on the pre-training phase. In this work, we consider a 4-layer SNN (i.e., L = 4).

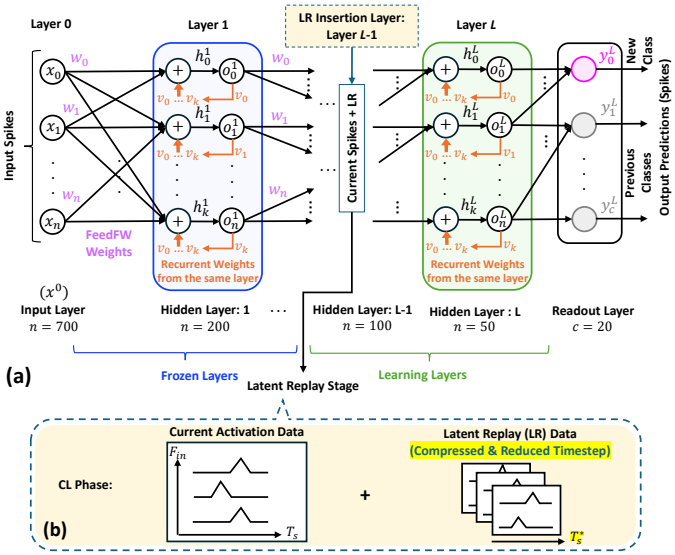


Fig. 6. Overview of (a) SNN architecture considered in this work, and (b) configuration of current activation data and latent replay (LR) data.

III. Proposed Replay4NCL Methodology

Our proposed methodology, illustrated in Fig. 4, employs several key steps, as outlined below to enable efficient memory replay-based NCL.

A. Perform Timestep Optimization

This step aims at optimizing the timestep setting for SNN processing, thereby substantially reducing the processing latency. To effectively perform such an optimization, we conduct an experimental case study that considers different timestep settings, and then analyze their impact on accuracy and latency. This way, we can identify the appropriate timestep settings that can provide high accuracy without any enhancements. Furthermore, to minimize the memory size of LR data (i.e., latent memory), we also employ a data compression-decompression mechanism based on the reduced timestep (see Fig. 7), thereby optimizing both the timestep and memory requirements while preserving information for NCL training phase.

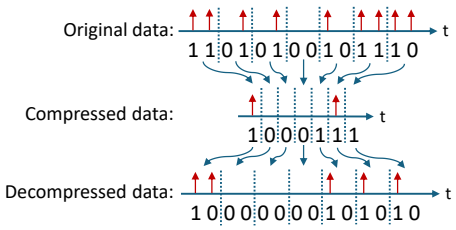


Fig. 7. A compression and decompression mechanism considered in this work; adopted from studies in [25].

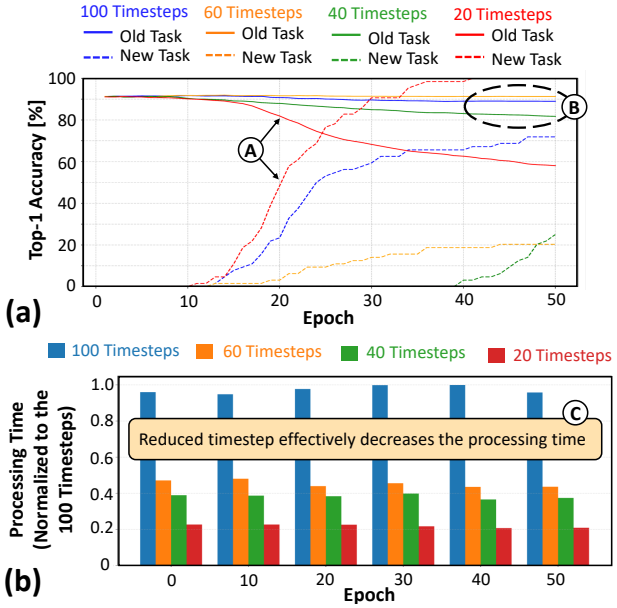


Fig. 8. Profiles of (a) accuracy and (b) processing latency across training epochs for different timestep settings (i.e., 100, 60, 40, and 20). Here, the timestep=100 represents the setting used by the state-of-the-art [25].

The experimental results are provided in Fig. 8, from which we draw several key observations below.

- Observation Ⓐ: An aggressive timestep reduction from 100 to 20 timesteps significantly reduces the accuracy of old tasks (old knowledge).
- Observation Ⓑ: To obtain acceptable accuracy for the old tasks (old knowledge) comparable to the state-of-the-art, we should employ at least 40 timesteps.
- Observation Ⓒ: Reduced timestep effectively decreases the processing time (latency), but they should be leveraged carefully to avoid significant accuracy degradation.

These observation points are leveraged for devising effective parameter adjustments in Section III-B.

B. Parameter Adjustments

This step aims at compensating the information loss due to the reduced timestep. Here, we observe that the generated spikes are less as compared to the original pre-trained model. The reason is that, neurons still employ the same threshold potential V_{thr} like the pre-trained model, but they encounter fewer number of spikes, which may not be enough for increasing the membrane potential V_{mem} to reach the threshold V_{thr}. Therefore, to make the network model adapt better to fewer spikes, we adjust the values of threshold potential V_{thr} and learning rate η with the following policy.

- **Threshold Potential (V_{thr}):** We adjust V_{thr} by reducing its value, since it will decrease the number of incoming spikes required to make V_{mem} reach V_{thr}, and hence preserving the spiking activity closer to the original pre-trained model.
- **Learning Rate (η):** We adjust η also by reducing its value, as it will decrease the learning speed of the network. This will ensure more careful weight updates during the training phase, considering that the network needs to leverage fewer spikes compared to the original pre-trained model.

C. Devise a Latent Replay Data Insertion Strategy

This step aims at devising a strategy to effectively benefit from the LR data insertion into the network. To achieve this, we perform a design exploration to see the impact of LR data insertion into different layers, and select the one that offers the highest accuracy. Additionally, this exploration also includes timestep optimization and parameter adjustments from Sections III-A and III-B. To incorporate all these techniques into an integrated mechanism, we propose a training strategy, whose key steps are presented in Alg. 1 and discussed below.

- First, we perform the pre-training phase, where the SNN model is trained and developed to learn all tasks from pre-training dataset; see Alg. 1: line 1-5.
- Second, we prepare the network for NCL training phase. To do this, we perform the following.
  - We generate the LR data activations (A_{LR}), and then split the SNN model based on the selected layer for LR data insertion (i.e., so-called *LR layer*), following the approach from [25]; see Alg. 1: line 6-20.
  - We propose to dynamically change the threshold potential V_{thr} based on the timestep, adjustment interval, and spike timing information. If the spikes occur during the defined interval, V_{thr} is increased; otherwise V_{thr} is decreased using a sigmoidal decay; see Alg. 1: line 10-17. This way, the generated A_{LR} adapt better to the reduced timestep.
- Third, we perform the NCL training phase by learning new tasks, while considering the following actions.
  - We dynamically adjust V_{thr} based on the timestep, adjustment interval, and spike timing information. This adaptive V_{thr} is updated based on the spike timing. If the spikes occur during the defined interval, V_{thr} is increased; otherwise V_{thr} is decreased; see Alg. 1: line 25-30.
  - We also reduce the learning rate η to make the network learn from fewer spikes more carefully; see Alg. 1: 21-33.

  As a result, the trained network is considered as the final output; see Alg. 1: 34-35.

IV. Evaluation Methodology

Fig. 9 illustrates the experimental setup for evaluating our Replay4NCL methodology. Here, we employ a Python-based implementation that runs on the Nvidia RTX 4090 Ti GPU machines with the SHD dataset [30] as the workload. For the network architecture, we use the SNN architecture shown in Fig. 6 with 4 layers. We also consider the class-incremental scenario, i.e., the SNN is trained with 19 tasks (classes) during the pre-training phase, and then it is trained with 20^{th} task

Algorithm 1 LR Data Training Strategy

Require: **[1]** Training set: pre-training (TS_{pre}), continual learning (TS_{cl}), subset of the pre-training (TS_{replay});
[2] Network parameters: network (net), number of layers (L), threshold potential (V_{thr}), learning rate (η), insertion layer for LR data (L_{ins}), reduced timesteps (T_{step}), adjust_interval=5;
[3] Epoch: pre-training (E_{pre}), continual learning (E_{cl});
[4] Activations: current (A_{new}), LR (A_{LR});
Ensure: Enhanced SNN model (SNN);

BEGIN
Pre-training phase:
Initialize SNN weights w and v randomly;
η_{pre} = 1e-3;
for (ep=0; $ep < E_{pre}$; ep++) **do**
  $w, v \leftarrow \text{train}_{pre}(net, (w, v), TS_{pre}, \eta_{pre}, V_{th})$;
end for
Network preparation for NCL training phase:
$\eta_{cl} = \eta_{pre}/100$;
$net_f, net_l = \text{split}(net, L_{ins})$;
for (ep=0; $ep < E_{pre}$; ep++) **do**
  for (t=0; $t < T_{step}$; t++) **do**
    if (t % adjust_interval == 0) **then**
      if any spikes occur **then**
        $avg_spike_time = \text{mean}(spike_timing)$;
        $V_{thr} = 1 + 0.01(T_{step} - avg_spike_time)$;
      end if
    else
      $V_{thr} = 1 \,/\, (1 + \exp(-0.001 \cdot t))$;
    end if
    $A_{LR} = \text{infer}(net_f, TS_{replay} \subseteq TS_{pre})$;
  end for
end for
NCL training phase:
$\eta_{cl} = \eta_{pre}/100$;
for (ep=0; $ep < E_{cl}$; ep++) **do**
  $A_{new} = \text{inference}\ (net_f, TS_{cl})$;
  for (t=0; $t < T_{step}$; t++) **do**
    if any spikes occur **then**
      $avg_spike_time = \text{mean}(spike_timing)$;
      $V_{thr} = 1 + 0.01(T_{step} - avg_spike_time)$;
    else
      $V_{thr} = 1 \,/\, (1 + \exp(-0.001 \cdot t))$;
    end if
    $w_l, v_l \leftarrow \text{train}(net_l, (w_l, v_l), A_{new} \cup A_{LR}, \eta_{cl}, V_{thr})$;
  end for
end for
$SNN \leftarrow net$;
return SNN;
END

(class) during the CL training phase. For the comparison partner, we consider the SpikingLR with its compressed LR data technique [25] as the state-of-the-art. The evaluation generates several results, including the accuracy for new and old tasks, latency, latent memory size, and energy consumption.

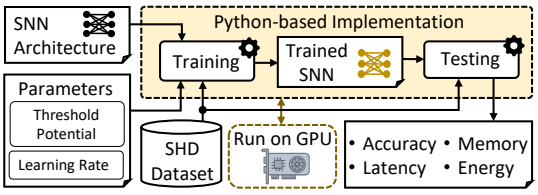


Fig. 9. Experimental setup considered in this work.

V. Experimental Results and Discussion

A. Maintaining High Accuracy for Old and New Tasks

Experimental results for accuracy across different LR insertion layers are shown in Fig. 10(a). In general, we observe that our Replay4NCL achieves comparable accuracy to the SpikingLR across different LR insertion layers for learning a new task. Moreover, for the LR insertion layers 0-2, our Replay4NCL achieves 100% accuracy when learning a new task within 50 epochs, which is better than the SpikingLR. For the LR insertion layer 3, the Replay4NCL achieves comparable accuracy to the SpikingLR when learning a new task (i.e., 67.19% for Replay4NCL and 71.88% for the SpikingLR); see ①. Meanwhile, when learning old tasks, our Replay4NCL achieves comparable high accuracy to the SpikingLR in the LR insertion layer 3 (i.e., 88.24% for Replay4NCL and 90.19% for the state-of-the-art); see ①. These results indicate that the Replay4NCL maintains high accuracy comparable to the SpikingLR on both learning the old and new tasks, despite employing low timestep settings. The reason is the Replay4NCL employs parameter adjustments to compensate the information loss due to fewer spikes, and leverages a training strategy that determines the effective threshold potential V_{thr} and learning rate η values.

Meanwhile, experimental results for accuracy across different training epochs considering the LR insertion layer 3, are provided in Fig. 11(a). We observe that Replay4NCL achieves higher Top-1 accuracy than the SpikingLR (see ④), since its parameter adjustments and training strategy effectively improve the learning process on the later layers.

B. Improvements on Processing Latency

Experimental results for processing time (latency) across different LR insertion layers are provided in Fig. 10(b). In general, our Replay4NCL significantly reduces the processing time as compared to the SpikingLR, i.e., by up to 2.34x speed-up; see ②. Similar trends are also observed for processing latency in the LR insertion layer 3 case, as shown by ⑤ in Fig. 11(b). These improvements are obtained mainly due to the reduction of timestep settings.

C. Latent Memory Savings

Experimental results for the latent memory size considering different LR insertion layers are provided in Fig. 12. In general, we observe that later layers tend to require smaller latent memory sizes due to smaller layer dimension (i.e., fewer number of neurons). Furthermore, these results also show that our Replay4NCL significantly saves the latent memory size as compared to the SpikingLR by 20%-21.88%, due to the timestep reduction which decreases the size of spike-based LR data activations to be stored in the memory. Such memory savings are beneficial for ensuring the applicability of our Replay4NCL for tightly-constrained embedded AI systems.

D. Reduction on Energy Consumption

Experimental results for energy consumption across different LR insertion layers are provided in Fig. 10(c). In general, we observe that our Replay4NCL significantly decreases the energy consumption from the SpikingLR across different LR insertion layers. Specifically, the Replay4NCL offers up to 56.7% energy savings; see ③. Similar trends are also observed for energy consumption in the LR insertion layer 3, as the Replay4NCL offers 36.4% energy saving; see ⑥ in Fig. 11(c).

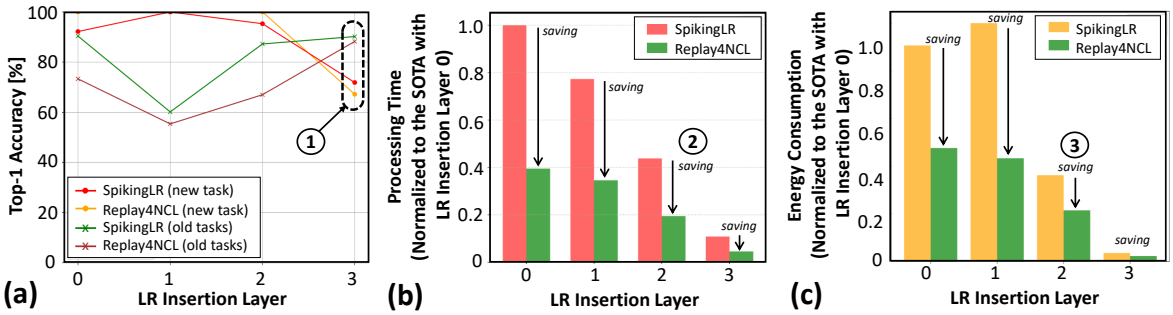


Fig. 10. Results for (a) Top-1 accuracy considering both old and new tasks, (b) processing time, and (c) energy consumption of the SpikingLR and our Replay4NCL across different LR insertion layers.

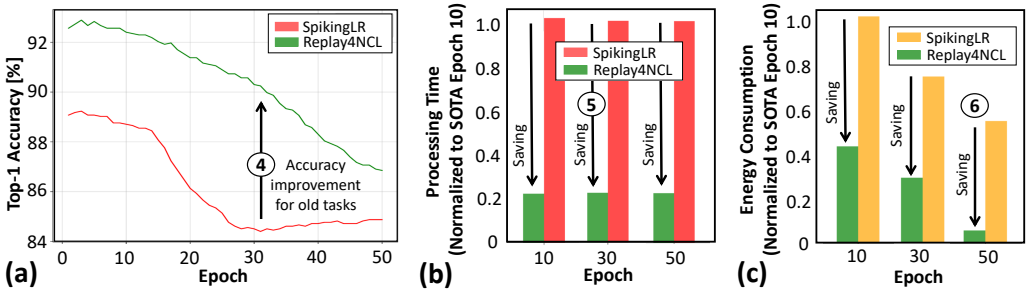


Fig. 11. Results for (a) Top-1 accuracy of old tasks, (b) processing time, and (c) energy consumption of the SpikingLR and our Replay4NCL across different training epochs considering the LR insertion layer 3.

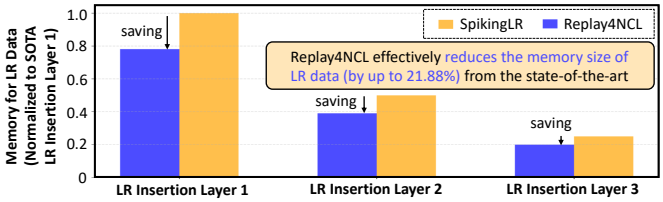


Fig. 12. Comparison of the latent memory sizes between the SpikingLR and our Replay4NCL across different LR insertion layers.

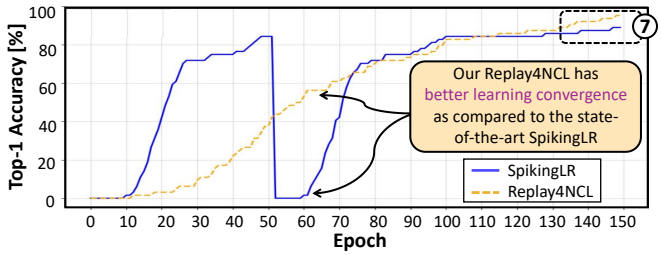


Fig. 13. Comparison of accuracy profiles between the SpikingLR and our Replay4NCL when learning a new task, while considering a long training period, i.e., 150 training epochs.

The reason is that, our Replay4NCL employs short timestep settings which significantly reduce the processing latency, thereby leading to low energy consumption.

E. Further Discussion

We further investigate the accuracy of the SpikingLR and our Replay4NCL considering a long training period (i.e., 150 epochs), and the experimental results are shown in Fig. 13. These results show that our Replay4NCL can achieve higher accuracy than the SpikingLR, as shown by ⑦. This indicates that the proposed LR training strategy in Replay4NCL provides an effective integrated mechanism to maximize the benefits from the proposed optimization through timestep reduction, and the NCL training enhancements through parameter adjustments. Moreover, with the employment of a lower learning rate η than the SpikingLR, our Replay4NCL offers more careful weight updates, which in turns leading to better learning convergence when processing fewer spikes as indicated by a smoother learning curve.

VI. Conclusion

In this paper, we propose the Replay4NCL methodology to enable efficient memory replay for NCL in embedded AI systems. Our Replay4NCL compresses LR data (old knowledge) and replays them during the CL training phase using small timestep settings, hence reducing latency and energy consumption. To compensate the information loss due to fewer spikes, we adapt the neuron threshold potential and learning rate settings. Experimental results on the class-incremental scenario with the SHD dataset demonstrate that Replay4NCL effectively supports the new task learning, while avoiding CF problem. Specifically, Replay4NCL maintains old knowledge with Top-1 accuracy of 90.43% compared to the state-of-the-art with 86.22%, while improving the latency by 4.88x, saving the latent memory by 20%, and saving the energy consumption by 36.43%. These results demonstrate the potential of our Replay4NCL methodology for enabling NCL for tightly-constrained embedded AI systems.

Acknowledgment

This work was partially supported by United Arab Emirates University (UPAR) with Fund code 12N169. This work was also partially supported by the NYUAD Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010.

REFERENCES

[1] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *International conference on machine learning*. PMLR, 2017, pp. 3987–3995.

[2] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2024.

[3] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3366–3385, 2021.

[4] R. V. W. Putra, M. A. Hanif, and M. Shafique, "Rescuesnn: enabling reliable executions on spiking neural network accelerators under permanent faults," *Frontiers in Neuroscience*, vol. 17, p. 1159440, 2023.

[5] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[6] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," vol. 24, pp. 109–165, 1989.

[7] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[8] G. M. van de Ven, N. Soures, and D. Kudithipudi, "Continual learning and catastrophic forgetting," *arXiv preprint arXiv:2403.05175*, 2024.

[9] A. Iyer, K. Grewal, A. Velu, L. O. Souza, J. Forest, and S. Ahmad, "Avoiding catastrophe: Active dendrites enable multi-task learning in dynamic environments," *Frontiers in neurorobotics*, vol. 16, p. 846219, 2022.

[10] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Advances in neural information processing systems (NIPS)*, 2017, pp. 4652–4662.

[11] R. V. W. Putra and M. Shafique, "Mantis: Enabling energy-efficient autonomous mobile agents with spiking neural networks," in *2023 9th International Conference on Automation, Robotics and Applications (ICARA)*, 2023, pp. 197–201.

[12] M. F. Minhas, R. V. W. Putra, F. Awwad, O. Hasan, and M. Shafique, "Continual learning with neuromorphic computing: Theories, methods, and applications," *arXiv preprint arXiv:2410.09218*, 2024.

[13] P. Panda, J. M. Allred, S. Ramanathan, and K. Roy, "Asp: Learning to forget with adaptive synaptic plasticity in spiking neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, vol. 8, no. 1, 2018.

[14] J. M. Allred and K. Roy, "Controlled forgetting: Targeted stimulation and dopaminergic plasticity modulation for unsupervised lifelong learning in spiking neural networks," *Frontiers in Neuroscience*, vol. 14, 2020.

[15] R. V. W. Putra and M. Shafique, "Spikedyn: A framework for energy-efficient spiking neural networks with continual and unsupervised learning capabilities in dynamic environments," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1057–1062.

[16] ——, "lpspikecon: Enabling low-precision spiking neural network processing for efficient unsupervised continual learning on autonomous agents," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.

[17] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan 2018.

[18] A. Ororbia, "Spiking neural predictive coding for continually learning from data streams," *Neurocomputing*, vol. 544, p. 126292, 2023.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[20] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[21] L. Pes, R. Luiken, F. Corradi, and C. Frenkel, "Active dendrites enable efficient continual learning in time-to-first-spike neural networks," *arXiv preprint arXiv:2404.19419*, 2024.

[22] B. Han, F. Zhao, Y. Zeng, W. Pan, and G. Shen, "Enhancing efficient continual learning with dynamic structure development of spiking neural networks," *arXiv preprint arXiv:2308.04749*, 2023.

[23] B. Han, F. Zhao, W. Pan, Z. Zhao, X. Li, Q. Kong, and Y. Zeng, "Adaptive reorganization of neural pathways for continual learning with hybrid spiking neural networks," *arXiv preprint arXiv:2309.09550*, 2023.

[24] M. Proietti, A. Ragno, and R. Capobianco, "Memory replay for continual learning with spiking neural networks," in *2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2023, pp. 1–6.

[25] A. Dequino, A. Carpegna, D. Nadalini, A. Savino, L. Benini, S. Di Carlo, and F. Conti, "Compressed latent replays for lightweight continual learning on spiking neural networks," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2024, pp. 240–245.

[26] D. Antonov, K. Sviatov, and S. Sukhov, "Continuous learning of spiking networks trained with local rules," *Neural Networks*, vol. 155, pp. 512–522, 2022.

[27] N. Skatchkovsky, H. Jang, and O. Simeone, "Bayesian continual learning via spiking neural networks," *Frontiers in Computational Neuroscience*, vol. 16, p. 1037976, 2022.

[28] M. Xiao, Q. Meng, Z. Zhang, D. He, and Z. Lin, "Hebbian learning based orthogonal projection for continual learning of spiking neural networks," *arXiv preprint arXiv:2402.11984*, 2024.

[29] R. V. W. Putra and M. Shafique, "Topspark: A timestep optimization methodology for energy-efficient spiking neural networks on autonomous mobile agents," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 3561–3567.

[30] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 7, pp. 2744–2757, 2020.

[31] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.

[32] J. D. Nunes, M. Carvalho, D. Carneiro, and J. S. Cardoso, "Spiking neural networks: A survey," *IEEE Access*, vol. 10, pp. 60 738–60 764, 2022.

[33] R. V. W. Putra and M. Shafique, "FSpiNN: An optimization framework for memory-efficient and energy-efficient spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 11, pp. 3601–3613, 2020.

[34] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Trans. on Neural Networks (TNN)*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004.

[35] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine (MSP)*, vol. 36, no. 6, pp. 51–63, 2019.

[36] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, vol. 12, 2018.

[37] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

NSFlow: An End-to-End FPGA Framework with Scalable Dataflow Architecture for Neuro-Symbolic AI

Hanchen Yang*[1], Zishen Wan*[1], Ritik Raj[1], Joongun Park[1], Ziwei Li[1], Ananda Samajdar[2], Arijit Raychowdhury[1], Tushar Krishna[1]
[1]*Georgia Institute of Technology, Atlanta, GA* [2]*IBM Research, Yorktown Heights, NY*

***Abstract*—Neuro-Symbolic AI (NSAI) is an emerging paradigm that integrates neural networks with symbolic reasoning to enhance the transparency, reasoning capabilities, and data efficiency of AI systems. Recent NSAI systems have gained traction due to their exceptional performance in reasoning tasks and human-AI collaborative scenarios. Despite these algorithmic advancements, executing NSAI tasks on existing hardware (e.g., CPUs, GPUs, TPUs) remains challenging, due to their heterogeneous computing kernels, high memory intensity, and unique memory access patterns. Moreover, current NSAI algorithms exhibit significant variation in operation types and scales, making them incompatible with existing ML accelerators. These challenges highlight the need for a versatile and flexible acceleration framework tailored to NSAI workloads.**

In this paper, we propose NSFlow, an FPGA-based acceleration framework designed to achieve high efficiency, scalability, and versatility across NSAI systems. NSFlow features a design architecture generator that identifies workload data dependencies and creates optimized dataflow architectures, as well as a reconfigurable array with flexible compute units, re-organizable memory, and mixed-precision capabilities. Evaluating across NSAI workloads, NSFlow achieves 31× speedup over Jetson TX2, more than 2× over GPU, 8× speedup over TPU-like systolic array, and more than 3× over Xilinx DPU. NSFlow also demonstrates enhanced scalability, with only 4× runtime increase when symbolic workloads scale by 150×. To the best of our knowledge, NSFlow is the first framework to enable real-time generalizable NSAI algorithms acceleration, demonstrating a promising solution for next-generation cognitive systems.

I. Introduction

Neuro-Symbolic AI (NSAI) emerges as a promising paradigm toward achieving artificial general intelligence (AGI) and human-like fluid intelligence. Compared to deep neural networks (DNNs), NSAI exhibits superior performance in cognitive tasks such as human-like learning, reasoning, and logical thinking [1]–[9]. NSAI synergistically combines neural approaches (e.g., DNNs) with symbolic representations (e.g., vectors, logics, graphs) to advance cognitive ability [10]–[16].

Despite its cognitive advantages, achieving real-time and efficient NSAI inference on resource-constrained devices presents significant challenges. These challenges stem from higher memory intensity, greater computational kernel heterogeneity, irregular memory access patterns, and underutilization of hardware resources. In our experiments, it takes >3 mins on NVIDIA desktop GPU to perform single reasoning task [17], underscoring the inefficiency of current solutions.

Previous work has identified three main challenges of NSAI [18]: *First*, *high memory footprint*. NSAI systems heavily rely on vector-symbolic architectures (VSAs) that use vector operations to encode symbolic knowledge, resulting in large memory footprints (often tens to hundreds of MB) and making it impractical to be fully cached on-chip in hardware accelerators. *Second*, *heterogeneous compute kernels*. Beyond neural networks, NSAI workloads incorporate diverse computations of varying sizes, such as vector convolutions, element-wise operations, and logical reasoning. These exhibit low data reuse, low compute array utilization, and limited parallelism, leading to inefficiencies on GPUs and TPUs. *Third*, *critical path dependency*. Symbolic reasoning often depends on outputs from neuro-perceptual modules, extending the critical path during cognitive inference and causing underutilization of traditional accelerators.

With the growing demand for scalable dataflow and architecture solutions for NSAI, FPGAs present an ideal platform due to their customizability, flexible memory management, and reconfigurability to adapt to evolving NSAI workloads. Previous work has demonstrated the potential of FPGAs for accelerating ML workloads [19]–[24]. However, FPGA deployment remains challenging for NSAI algorithms due to the complexity of organizing on-chip resources and limited memory capacity [25]–[27].

To address these challenges, we identify unique opportunities to enhance NSAI acceleration efficiency and propose NSFlow, a scalable FPGA-based dataflow architecture design automation framework. To the best of our knowledge, NSFlow is the *first* automated end-to-end solution for accelerating and deploying generic NSAI workloads. NSFlow features a frontend subsystem with *dataflow architecture generator* that includes NSAI execution trace extraction, dataflow graph generation, and a two-phase design space co-exploration strategy, and a backend subsystem with *flexible neuro-symbolic hardware architecture* with adaptive array folding, reconfigurable memory partitioning, and efficient heterogeneous storage. By integrating its frontend and backend, NSFlow delivers efficient and scalable acceleration for NSAI workloads. Specifically, it identifies data dependencies, explores design space options, and generates optimized dataflow architectures tailored for FPGA deployment.

This paper, therefore, makes the following contributions:

1) An end-to-end FPGA design automation framework for accelerating and deploying generic NSAI workloads.
2) A design generator that (i) identifies workload-specific data dependencies using a self-generated dependency graph tailored for vector-symbolic-based NSAI algorithms and (ii) derives an optimal dataflow architecture through a novel design space co-exploration strategy.
3) A hardware architecture featuring a flexible neuro-symbolic systolic array, an efficient SIMD unit, reorganizable on-chip memory, and support for mixed-precision computations.

II. Neuro-Symbolic AI and Characterization

This section presents NSAI algorithms with key kernels (Sec. II-A), and analyzes their workload characteristics (Sec. II-B).

A. Neuro-Symbolic AI Algorithm

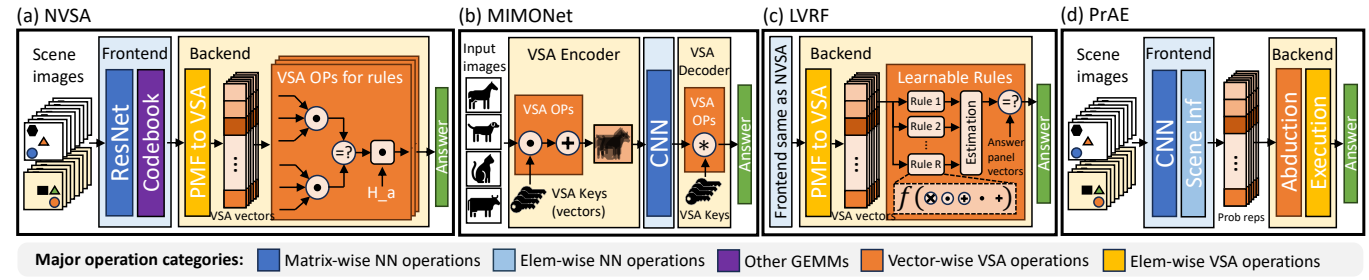
Neurosymbolic AI synergistically integrates *learning capability* of neural networks with *reasoning capability* of symbolic AI, offering data-efficient learning and transparent, logical decision-making beyond traditional DNNs. ❶ **Neural system.** The process begins with a neural module that handles perception tasks by interpreting sensory data and generating meaningful scene and object representations,

*Equal Contributions. (hanchen@gatech.edu, zishenwan@gatech.edu)
This work was supported in part by CoCoSys, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation program sponsored by DARPA.

TABLE I

NEUROSYMBOLIC MODELS. SELECTED NEUROSYMBOLIC AI WORKLOADS FOR ANALYSIS, REPRESENTING A DIVERSE OF APPLICATION SCENARIOS.

Representative Neuro-Symbolic AI Workloads		Neuro-Vector-Symbolic Architecture (NVSA) [17]	Multiple-Input-Multiple-Output Neural Networks (MIMONet) [28]	Probabilistic Abduction via Learning Rules in Vector-symbolic Architecture (LVRF) [12]	Probabilistic Abduction and Execution Learner (PrAE) [5]
Compute Pattern	Neuro	CNN	CNN/Transformer	CNN	CNN
	Symbolic	VSA binding/unbinding (Circular Conv)	VSA binding (Circular Conv)	VSA binding/unbinding (Circular Conv)	Probabilistic abduction
Application Scenario	Use Case	Spatial-temporal and abstract reasoning	Multi-input simultaneously processing	Probabilistic reasoning, OOD data processing	Spatial-temporal and abstract reasoning
	Advantage vs. Neural	Higher joint representation efficiency, Better reasoning capability, Transparency	Higher throughput, Lower latency, Compositional compute, Transparency	Stronger OOD handling capability, One-pass learning, Higher flexibility, Transparency	Higher generalization, Transparency, Interpretability, Robustness



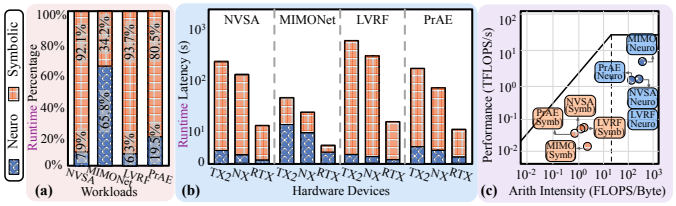


Fig. 1. **End-to-end neuro-symbolic runtime and roofline characterization.** **(a)** Benchmark neuro-symbolic models on CPU+GPU system, showing symbolic may serve as system bottleneck. **(b)** Benchmark on Coral TPU, TX2, NX, and 2080Ti GPU, showing that real-time performance cannot be satisfied. **(c)** Roofline of RTX 2080Ti GPU, indicating symbolic memory-bounded.

providing essential inputs for reasoning. ❷ **Symbolic system.** These features are then passed to the symbolic system for reasoning, enhancing explainability and reducing reliance on extensive training data by leveraging established models of the physical world (e.g., rules and coded knowledge). This step integrates learned neural network knowledge with symbolic rules, allowing the system to both learn from new data and reason logically based on existing knowledge. The outputs of symbolic reasoning are used for decision-making, and response or action generation.

Tab. I highlights four representative neuro-symbolic workloads: NVSA [17] for spatial-temporal reasoning, MIMONet for multi-input processing [28], LVRF for probabilistic abduction [12], and PrAE for abstract reasoning tasks [5]. These workloads demonstrate superior reasoning capabilities and represent a promising paradigm for human-like intelligence. These approaches integrate CNNs for neuro and vector-symbolic architectures (VSAs) for symbolic processing.

A key VSA operation is the blockwise *circular convolution* that combines two vectors in a way that preserves the information from both, making it suitable for representing composite symbols. Mathematically, the circular convolution of two vectors $\mathbf{A}$ and $\mathbf{B}$ (each of dimension N) generates vector $\mathbf{C}$ as $C[n] = \sum_{k=0}^{N-1} A[k] \cdot B[(n-k) \bmod N]$ where each element of $\mathbf{C}$ is obtained by multiplying the elements of $\mathbf{A}$ with the circularly shifted elements of $\mathbf{B}$, and then summing up. Circular convolution has commutativity and associativity properties, making it particularly effective in hierarchical reasoning tasks where manipulating structured information is critical.

B. Neuro-Symbolic AI Workload Characterization

To understand the real-device efficiency of neuro-symbolic AI workload, recent work [29] profiles four representative models as

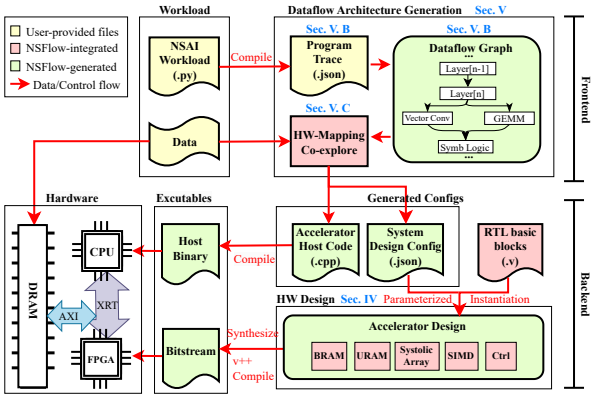


Fig. 2. **NSFlow Overview.**

elaborated in Tab. I on Coral edge TPU (4 W), Jetson TX2 (15 W), Xavier NX (20 W), and RTX 2080Ti (250 W), respectively.

End-to-end latency breakdown. Fig. 1a and Fig. 1b present the end-to-end latency breakdown of neuro-symbolic workloads, highlighting three key observations: (1) *The real-time performance cannot be satisfied* across devices. Even with additional compute resources to reduce NN runtime, the substantial overhead from symbolic reasoning prevents real-time execution. (2) *Symbolic operations dominate runtime.* For instance, symbolic modules account for 87% of NVSA total runtime while contributing only 19% of its total FLOPS, suggesting that symbolic operations are not efficiently handled by GPUs/TPUs. (3) *Symbolic reasoning computation lies on the critical path* as its computation depends on outputs from the neural modules.

System Roofline Analysis. Fig. 1c employs the roofline model of RTX 2080Ti GPU version to quantify the neurosymbolic workloads. We observe that *symbolic modules are memory-bounded while neuro modules are compute-bounded*. This is mainly due to symbolic operations requiring streaming vector elements, increasing the memory bandwidth pressure and resulting in hardware underutilization.

III. NSFLOW OVERVIEW

NSFlow is an end-to-end framework that identifies data dependency, explores the design space, and generates an optimal dataflow architecture design for FPGA deployment tailored to a given NSAI workload. Fig. 2 (a) shows an overview of the proposed framework, divided into frontend and backend.

A. NSFlow Frontend

The *Design Architecture Generator (DAG)* is the core component of the frontend operating on the host side. The DAG module begins by extracting an execution trace from the user-provided workload. It then generates a dataflow graph specifically designed for VSA-based NSAI workloads, capturing operator-level specifications, runtime, memory functions, and their data dependencies. This dataflow graph is used for co-exploration of dataflow architecture through a novel two-phase algorithm. The first phase identifies the optimal system design configuration for the FPGA accelerator, while the second determines an efficient (or near-optimal) reconfiguration and mapping scheme. These configurations are specified in the accelerator host code, enabling the CPU to invoke device kernels via the XRT API. After compilation, the CPU executes the host binary to schedule operations on the FPGA.

B. NSFlow Backend

The NSFlow backend includes a pre-define accelerator template comprising several essential components: BRAM blocks for flexible on-chip memory, adaptive Systolic Array for parallel neuro and symbolic operations, SIMD unit for element-wise, vector reduction and scalar operations, and control logic for task scheduling on hardware level. These components are parameterized using the system design configuration file generated by the frontend, enabling the instantiation of an optimized microarchitecture based on workload characterization. NSFlow then synthesizes and compiles the RTL into an executable bitstream for deployment. During real-time inference, the CPU executes the host binary code to run FPGA kernels and manages off-chip memory transactions through AXI interfaces.

We will present the NSFlow design in a bottom-up manner, starting with the backend flexible neuro-symbolic hardware architecture (Sec. IV) and followed by the frontend graph and dataflow architecture generators (Sec. V).

IV. NSFlow Backend: Flexible Hardware Architecture

This section first presents an overview of the NSFlow hardware architecture (Sec. IV-A), then walks through our design featuring an *NS-adaptive systolic array* (Sec. IV-B), *Re-organizable on-chip memory* (Sec. IV-C), *Adaptive mixed precision computation* (Sec. IV-D), an *Efficient custom SIMD unit* (Sec. IV-E).

A. Overview of NSFlow Hardware Architecture

Fig.3(a) exhibits the hardware architecture of NSFlow. It consists of a uniquely designed NSAI-workload-adaptive Systolic Array for NN and Vector-symbolic operations, a SIMD unit for reductions, element-wise operations, flexibly arranged on-chip RAM blocks, and a control unit for kernel scheduling and memory transactions. NSFlow has pre-defined RTL of all the above blocks with scaling parameters subject to the design configuration generated from DAG for optimal execution.

B. Adaptive Systolic Array (AdArray)

Inspired by [29], we implement an adaptive systolic array design (AdArray) to maximize efficiency for NSAI inference.

Adaptive array folding. AdArray can run both NN ops and vector-symbolic circular convolution, the two most dominating components in our targeted NSAI workload, on its arbitrary portions (sub-arrays) simultaneously to maximize parallelism and utilization. Each sub-array is either combined with its adjacent one to operate NN ops, or singularly running vector operations like circular convolution in the symbolic binding process. In Fig. 3(a) we showcase the design with a mini 4×6 systolic array, which is split into 3 sub-arrays with each ranging 2 columns (c0 and c1 for A1, c2 and c3 for A2, c4 and c5 for A3). In this case, A1 and A2 are combined together to perform NN ops, while in A3 each column is running vector-symbolic operations.

Efficient vector-symbolic circular convolution streaming. Traditional TPU's systolic array is extremely inefficient for circular convolution operations with heavy memory transactions and low parallelism due to non-ideal spatial and temporal mapping. Fig. 3(b) showcases how a single column in our design performs vector-symbolic operation with a 3-element circular convolution example. The first vector $\boldsymbol{A}$ is held in stationary registers, while the second vector $\boldsymbol{B}$ is streamed from SRAM. The MAC unit processes the data from both stationary and streaming registers, adding it to the partial product received from the PE above. A passing register temporarily stores the streaming input for a cycle before it moves to the streaming register. This value is transferred to the passing register of the next PE in the following cycle. The procedure is repeated until the final circular convolution outputs. Unlike traditional Systolic Arrays, each PE uses an extra register named Passing Reg at the top of one of the input ports to cause a 1-cycle streaming pace mismatch between the two input vectors $\boldsymbol{A}$ and $\boldsymbol{B}$, thus enabling circular convolution operations. Note that to enable this type of streaming, each PE needs to have one extra vertical input port connected with its above PE's previous right output port as depicted in Fig. 3(b). When performing NN operations, the Passing Register is bypassed via multiplexer, and the horizontal connections in the sub-array are again established to enable weight and input passing as in a traditional Systolic Array.

Two-level flexibility. Our efficient systolic array design also benefits from extraordinary flexibility at both design level and kernel level. At design level, our DAG decides the array size and its memory size (Sec. IV-C), as well as the number of sub-arrays that best fits the overall workload characteristic (Sec. V); At kernel level, the array are reconfigured to an optimal folding scheme at runtime, maximizing utilization and parallelism for the NN and vector-symbolic operations.

C. Re-Organizable On-Chip Memory

The profiled NSAI workloads feature heavy memory usage and versatile computing kernels, thus we design a flexible memory system to enable smooth executions and transactions with limited FPGA on-chip memory resource (∼36MB on ZCU104), which features ❶ **Re-organizable memory partition**, ❷ **Adaptive memory size**, and ❸ **Efficient heterogeneous storage** to fully exploit FPGA's potential and maximize memory efficiency for NSAI workloads.

As shown in Fig. 3, the on-chip memory system consists of three memory blocks Mem_A, Mem_B, and Mem_C, an on-chip cache, and a memory bus for off-chip transactions. Mem_A, Mem_B, and Mem_C are all double-buffered memories to enable seamless read and write among off-chip memory and the systolic array. ❶ Mem_A is partitioned into two chunks - Mem_{A1} and Mem_{A2} - to simultaneously load NN layers and vector data for the corresponding sub-array in AdArray . When performing NN operations or vector operations singularly, the two memory chunks can be merged into one at runtime for better performance and simpler control. $MemB$ works as the IFMAP buffer in normal systolic arrays which feeds data to the horizontal inputs of AdArray only for NN processing. Mem_C stores the outputs from AdArray and the SIMD unit which are either read by the compute units, or written to Mem_A/Mem_B or off-chip DRAM. The on-chip cache buffers intermediate results for the 3 memory blocks. ❷ The sizes of all above memory components will be defined by DAG based on workload's characteristics and dataflow. ❸ In real FPGA deployment, Mem_A, Mem_B, and Mem_C are comprised of

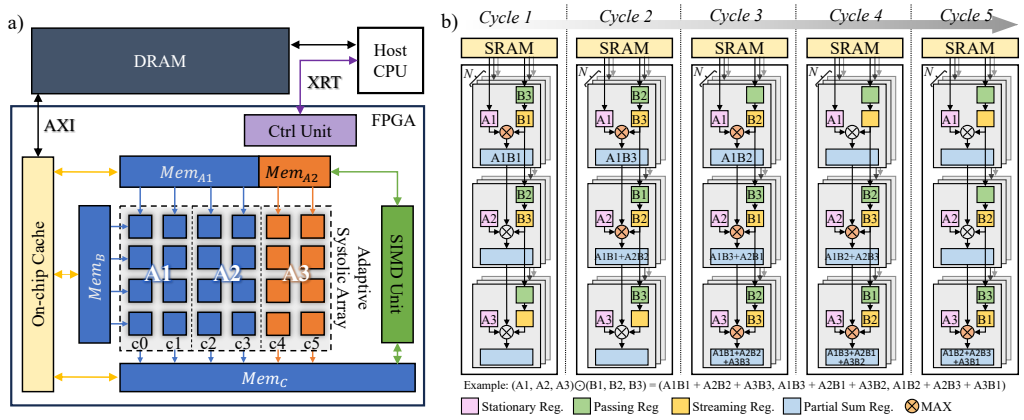


Fig. 3. **NSFlow Hardware Architecture.**

numerous 18KB BRAM blocks for maximum configurability, and on-chip cache is built with URAM considering its large capacity (288KB per block). Small registers and buffers in compute element use LUTRAMs for fast and dynamic access.

D. Adaptive Compute for Mixed Precision

To improve computing efficiency and save on-chip memory usage, NSFlow supports mixed precisions ranging from FP16/8 to INT8/4 in different components of the workload specified by user at frontend. DAG employs compute units adaptive to various precisions. In NVSA for example, NN and Symbolic operations are quantized to INT8 and INT4 respectively, thus the multipliers in AdArray and the SIMD support both precisions with sufficient leverage of DSP units [30]. Low-precision additions are handled by LUT for fast outcome.

E. Efficient Custom SIMD Unit

NSFlow incorporates a custom SIMD unit to efficiently perform vector reductions, element-wise operations, etc., with fluid data transfer between the output of the NSFlow array and the input SRAM for successive executions. It comprises multiple processing elements (PEs), each equipped with compact logic circuits (i.e., sum, mult/div, exp/log/tanh, norm, softmax, etc.) to handle vector operations or optimized sparse computations on mixed level of quantized data.

V. NSFlow Frontend: Dataflow Architecture Generation

In the frontend, we implement a *Dataflow Architecture Generator (DAG)* that first builds a dataflow graph based on operation trace extracted from the workload, then generates an optimal (or sub-optimal) dataflow architecture design, defined by a design configuration file for instantiating hardware modules, and a host code for the CPU to schedule accelerator kernels.

This section first identifies the NSAI dataflow challenges (Sec. V-A). Then we illustrates the process to generate *operation graph* and subsequently *dataflow graph* (Sec. V-B). Finally, we discuss how *DAG* searches for an optimal architectural design and mapping based on the *dataflow graph* (Sec. V-C).

A. NSAI Dataflow Challenges

We identify three main NSAI dataflow challenges (Fig. 4(b)). First, the sequential execution and frequent interactions of neural and symbolic components results in increased latency and low system throughput. Second, the heterogeneous neural and symbolic kernels lead to low compute array utilization and efficiency of ML accelerators. Third, heavy memory transactions exhibited in both components can cause large communication latency, which is even more challenging in FPGA deployment.

DAG perfectly addresses the above challenges by first, identifying data dependencies through dataflow graph to fully exploit parallelism opportunities among NN and symbolic operations with featured HW architecture; second, balancing NN and symbolic operations on our AdArray , empowered by both *design-level flexibility* and *kernel-level flexibility* to achieve maximum utilization; third, configuring memory units adaptively to best-fit workload's memory usage, thus eliminating unnecessary transactions and stalls.

B. Data Dependency Identification

```
graph():
    ...
    // Neuro Operation - CNN (Resnet18)
    %relu_1[16,64,160,160] : call_module[relu](args = (%bn1
        [16,64,160,160]))
    %maxpool_1[16,64,160,160] : call_module[maxpool](args =
        (%relu_1[16,64,160,160]))
    %conv2d_1[16,64,160,160] : call_module[conv2d](args =
        (%maxpool_1[16,64,160,160]))
    ...
    // Symbolic Operations
    // Inverse binding of two block codes vectors by
        blockwise cicular correlation
    %inv_binding_circular_1[1,4,256] : call_function[nvsa.
        inv_binding_circular](args = (%vec_0[1,4,256], %
        vec_1[1,4,256]))
    %inv_binding_circular_2[1,4,256] : call_function[nvsa.
        inv_binding_circular](args = (%vec_3[1,4,256], %
        vec_4[1,4,256]))
    // Compute similarity between two block codes vectors
    %match_prob_1[1] : call_function[nvsa.match_prob](args
        = (%inv_binding_circular_1[1,4,256], %vec_2
        [1,4,256]))
    // Compute similarity between a dictionary and a batch
        of query vectors
    %match_prob_multi_batched_1[1]: call_function[nvsa.
        match_prob_multi_batched](args = (%
        inv_binding_circular_2[1,4,256], %vec_5[7,4,256]))
    %sum_1[1] : call_function[torch.sum](args = (%
        match_prob_multi_batched_1[1]))
    %clamp_1[1] : call_function[torch.clamp](args = (%sum_1
        [1]))
    %mul_1[1] : call_function[operator.mul](args = (%
        match_prob_1[1], %clamp_1[1]))
    ...
```

Listing 1. Neuro-Vector-Symbolic Architecture Profiling Result

Program trace. NSFlow first extracts an execution trace from input program through compilation, and pre-process the file to be ready for later dataflow graph generating. Listing 1 exhibits a snapshot taken from NVSA program trace, with representative kernels for Neural and Symbolic parts showing data dependencies.

Dataflow Graph. Fig. 4 depicts how a *Dataflow Graph* is derived. ❶ **Critical path identification:** It begins with a DFS through the execution graph previously generated, identifying the critical

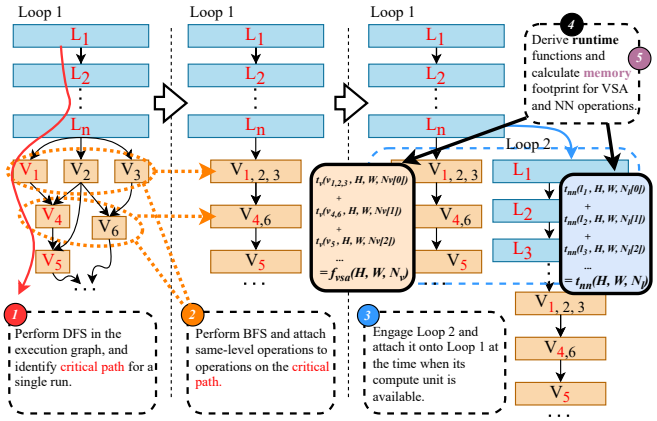


Fig. 4. **Dataflow Architecture Generation (DAG) Flow.**

TABLE II

NSFLOW DESIGN SPACE. Maximum #PEs = 2^m. With *exploration phasing* and *space pruning*, search space is reduced by 100 magnitudes

	HW config (H, W, N)	Array partition and mapping	Total design space, $m = 10$
Original	$m \times (m+1)/2$	$(N-1)^k$ for each N	10^{300}
DAG	*PhaseI*: $1/4 \le H/W \le 16$	*PhaseII*: Iter × #layers	10^3

path for a single loop of the workload. ❷ **Inner-loop parallelism identification:** Then DAG walks through the graph again with BFS, to identify operation nodes at the same depth as the nodes on the critical path, and attach them to the corresponding critical-path nodes, indicating their earliest execution and parallelisms. For a single loop, NN layers are typically on the critical path without any attached nodes due to its strict dependencies between layers, while symbolic parts may have more parallelism opportunities thus more grouped nodes. ❸ **Inter-loop parallelism identification:** After reshaping the graph for a single loop, DAG attaches the next loop's graph to the existing one by positioning the first operation of the second loop at the time its required computing unit is freed. For example in the third step shown in Fig. 4(b), the first NN layer (L_1) in Loop 2 starts as soon as the last NN layer (L_n) of Loop 1 finishes and runs along with the symbolic operations in Loop 1. ❹ **Runtime function derivation:** For each node in the newly fused graph, DAG derives their runtime functions (Sec. V-C) on corresponding sub-arrays with operation parameters (i.e. vector quantity n and dimension d, NN layer dimensions in m, n, k, etc.) and configuration variables (i.e. sub-array's height, width and partition scheme). ❺ **Memory cost calculation:** DAG also computes memory footprint based on each node's data size for later memory block configuring.

Dataflow Graph describes data dependencies, inner/inter-loop parallelisms as well as their runtime and memory cost model for real-time execution on AdArray. Next we discuss how DAG uses it to explore the HW design and mapping.

C. Two-Phase Design Space Exploration

Design Space. NSFlow's adaptive architecture (Sec. IV) and the Dataflow Graph (Sec. V-B) creates a large cross-coupled design space, defined by the hardware configuration with height (H), width (W) and number (N) of the sub-arrays, and the mapping scheme specified by the number of sub-arrays running NN layers and VSA operations (i.e. $N_l[i]$ for layer node i and $N_v[j]$ for VSA node j, where), which could vary for each node in the Dataflow Graph during runtime. Note that N_l and N_v are both vector variables with lengths equal to the number of layer and VSA nodes in a single loop. The total size of this design space reaches 10^{300} (Tab. IV), making brutal force search impractical. Next we present how we derive runtime function for the nodes and our unique DSE algorithm.

Algorithm 1: NSFlow Two-Phase DSE Algorithm

Data: R_l, R_v, $Range_H$ (H search range), $Range_W$ (W search range), M (max #PEs), $Iter_{max}$ (Phase II max iterations)
Result: H, W, N (total #sub-arrays), N_l, N_v

```
/* Phase I */
for H in Range_H, W in Range_W do
    N = ⌊M/(H × W)⌋ // get total #sub-arrays
    for N̄_l in [1, N) do
        // get optimal HW config for parallel mapping
        Set all elements in N_l to N̄_l
        Set all elements in N_v to N − N̄_l
        t_para = max(t_nn(H, W, N_l), t_vsa(H, W, N_v))
        Save the H, W, N̄_l (and N̄_v) with minimal t_para.
    end
    // get sequential runtime
    t_seq = Σ_i^G f_{l_i}(H, W, N) +
      min(Σ_j^G f_{v_j,temp}(H, W, N), Σ_j^G f_{v_j,spatial}(H, W, N))
    // Set to sequential mode in case it has better performance
    Return and set sequential mode if t_seq < t_para else Continue
end
/* Phase II */
for it in Iter_max do
    for layer i in R_l do
        Locate VSA node j' and j'' where layer i starts and ends
        if t_seq < t_para do N_l[i] − −; N_v[j' : j''] + +;
        else do N_l[i] + +; N_v[j' : j''] − −;
        t_para = max(t_nn(H, W, N_l), t_vsa(H, W, N_v))
        Save the H, W, N_l, N_v with minimal t_para.
    end
end
Return H, W, N, N_l, N_v.
```

Analytical models. Inspired by the analytical models from previous research [29], [31], we derive runtime functions specifically for NSFlow. Since AdArray is a scale-out design with row-level partition, the NN runtime for layer node i can be calculated as:

$$t_l(H, W, N_l[i]) = (2H + W + d_{1,i} - 2) \times \lceil \lceil d_{2,i}/N_l[i] \rceil / H \rceil \times \lceil d_{3,i}/W \rceil \quad (1)$$

where H and W are sub-array height and width. $d_{1,i}$, $d_{2,i}$ and $d_{3,i}$ are layer dimensions m, n, k. Assuming R_l is a set collecting all layer nodes within a loop, NN total runtime is:

$$t_{nn}(H, W, N_l) = \Sigma_i^{R_l} t_l(H, W, N_l[i]) \quad (2)$$

For a VSA node j, runtime for spatial and temporal mapping are respectively:

$$t_{v,spatial}(H, W, N_v[j]) = n_j \times \lceil d_j/(W \times H \times N_v[j]) \rceil \times T \quad (3)$$

$$t_{v,temp}(H, W, N_v[j]) = \lceil n_j/W \rceil \times \lceil d_j/H \times N_v[j] \rceil \times T \quad (4)$$

where $T = 3 \times H + d_j - 1$, n_j and d_j are the vector quantity and size respectively. DAG uses the fastest mapping scheme, so with R_v as the set of all VSA nodes, the total VSA runtime in a single loop is:

$$t_{vsa}(H, W, N_v) = min(\Sigma_j^{R_v} t_{v,temp}(H, W, N_v[j]), \Sigma_j^{R_v} t_{v,spatial}(H, W, N_v[j])) \quad (5)$$

AdArray Design Generation. We describe how DAG generates AdArray design and mapping scheme in Algorithm 1. To mitigte the search space, the DSE process is decoupled into 2 phases, exploiting AdArray's two-level flexibility (Sec. IV-B):

In Phase 1, DAG assumes static partitions among nodes to limit search space(i.e. $\forall i, j, N_l[i] = \bar{N_l}$, $N_v[j] = \bar{N_v}$). It finds the optimal H, W, N constrained by maximum number of PEs M defined based on FPGA resource, and a fixed partition scheme defined by $\bar{N_l}$ and $\bar{N_v}$ to maximizes overall performance. The search space of H, W, N is also pruned based on analytical model results. Phase 2 explores the

TABLE III
DESIGN CONFIGURATION AND FPGA DEPLOYMENT.

Workloads	Precision		AdArray Configuration		SIMD Size	On-chip SRAM Blocks (BRAM)			On-chip Cache (URAM)	AMD U250 Utilization						Frequency
	NN	Symb	Size (H, W, N)	Default Partition ($\bar{N}_l : \bar{N}_v$)		MemA1, MemA2	Mem B	Mem C		DSP	LUT	FF	BRAM	URAM	LUTRAM	
NVSA	INT8	INT4	32, 16, 16	14 : 2	64	2.7 MB, 1.1 MB	2.7 MB	1.6 MB	16.2 MB	89%	56%	60%	34%	8%	24%	272 MHz
MIMONet	INT8	INT8	32, 32, 8	6 : 2	64	3.4 MB, 1.2 MB	3.4 MB	2.1 MB	20.1 MB	89%	44%	52%	43%	10%	20%	272 MHz
LVRF	INT8	INT4	32, 16, 16	14 : 2	64	2.7 MB, 0.96 MB	2.7 MB	1.4 MB	15.5 MB	89%	56%	60%	31%	7%	24%	272 MHz

TABLE IV
NSFLOW ALGORITHM OPTIMIZATION PERFORMANCE. NSFlow exhibits comparable reasoning capability with the proposed mixed precision.

Reasoning Accuracy	FP32	FP16	INT8	MP (IN8 for NN, INT4 for Symb)	INT4
RAVEN [32]	98.9%	98.9%	98.7%	98.0%	92.5%
I-RAVEN [33]	99.0%	98.9%	98.8%	98.1%	91.3%
PGM [34]	68.7%	68.6%	68.4%	67.4%	59.9%
Memory	32MB	16MB	8MB	5.5MB	4MB

mapping scheme further by efficiently fine-tuning N_l and N_v around $\bar{N}_l$ and $\bar{N}_v$ in the dataflow loop with maximum number of iterations pre-defined as $Iter_{max}$, seeking optimal or near-optimal partitions. Searching granularity is set to the span of each NN layer, as VSA kernels are in general smaller and more flexible to be fit into arbitrary array shapes. With $O(N)$ complexity, our two-phase DSE algorithm shrinks the design space by $10^{100}\times$ shown in Tab. IV.

Memory and SIMD unit. After generating the design of AdArray , memory blocks sizes are computed based on node memory cost to eliminate inner-node memory stalls, for example $M_{A1} = max(filter\ size\ in\ R_l)$, $M_{A2} = max(node\ size\ in\ R_v)$. M_{A1} and M_{A2} are merged for non-parallel operations. On-chip cache size is $2\times(M_A + M_B + M_C)$. SIMD size is minimized such that latency of concurrent elem-wise/vector reduction operations can be hidden.

Unlike other DSE work [35], [36] that only focuses on single task mapping on traditional systolic array, NSFlow exploits NSAI inter-task and inner-task parallelism opportunities on AdArray at both hardware level and mapping level, boosting the performance of versatile NSAI workloads.

VI. EVALUATION RESULTS

A. Experimental Setup

Algorithm setup. We evaluate NSFlow with three state-of-the-art VSA-based NSAI workloads, i.e., NVSA [17], MIMONet [28], and LVRF [12] on the commonly-used spatial-temporal reasoning datasets - RAVEN [32], I-RAVEN [33], PGM [34], CVR [37], and SVRT [38]. Following [12], [17], [28], we select the training hyperparameters based on the end-to-end reasoning performance on the validation set.

Hardware setup. We consider several hardware baselines, including TX2, Xavier NX, Xeon CPU, RTX 2080, and ML accelerators (TPU, Xilinx DPU). NSFlow framework can be deployed on any type of FPGA board. Tab. III showcases our deployment for 3 algorithms on AMD U250 using Xilinx Vivado and Synopsys Design Compiler. The clock frequency is set to 272MHz.

B. NSFlow Performance

Mixed-precision performance. We benchmark NSAI model on three spatial-temporal reasoning datasets to first evaluate the effectiveness of mixed quantization in NSFlow. As shown in Tab. IV, we can observe that NSFlow mixed precision achieves comparable accuracy with NVSA algorithm [17] while with 5.8× memory footprint savings. Similar results are observed in MIMONet/LVRF on CVR/SVRT datasets. It is also worth noting that neurosymbolic methods consistently achieve improved cognition and reasoning capability than neural network-based methods and surpass human performance.

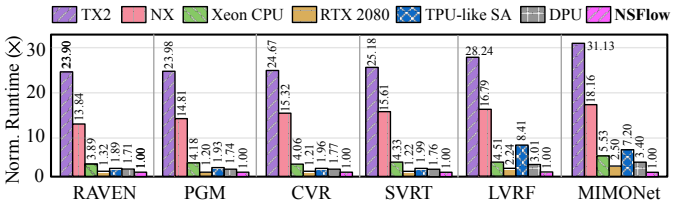


Fig. 5. **End-to-End Runtime Improvement.** NSFlow consistently outperforms Xilinx DPU, TPU-like accelerator, Xeon CPU, RTX GPU, and edge SoCs (TX2, NX) in end-to-end runtime evaluated on NSAI reasoning tasks.

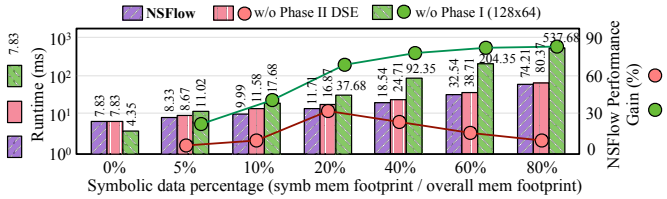


Fig. 6. **Ablation Study.** NSFlow exhibits superior scalability comparing to normal TPU design across workloads with various symbolic proportions.

Performance improvement. We first benchmark our NSFlow accelerator against edge SoC (Jetson TX2, Xavier NX), Intel Xeon CPU, Nvidia RTX 2080, TPU-like systolic array (128×128), and Xilinx DPU. For accelerating NSAI algorithm on six reasoning tasks featuring different difficulties. We can observe in Fig. 5 that NSFlow accelerator consistently outperforms other devices, offering 31×/18× speedup over TX2 and NX, more than 2× over GPU, up to 8× speedup over TPU-like systolic array, and more than 3× speedup over Xilinx DPU on some standard workloads.

Ablation study. To further showcase the scalability of NSFlow and validate the necessity of proposed DSE algorithm, in Fig. 6 we summarize the runtime of an NSFlow-generated architecture (32×32×8) w/ and w/o the proposed mapping and hardware techniques, evaluated on a NVSA-like workload with varying vector-symbolic data proportions alongside a ResNet18. We observe that despite slight overhead caused by array partition when symbolic part is minimal ($< 1\%$), (1) with symbolic ratio going up NSFlow speedup against traditional systolic array grows steadily, reaching up to more than 7× when symbolic data occupies 80% of the memory. (2) The performance gain from our two-phase DSE algorithm (compared to only having array folding, or Phase I) can reach 44% when symbolic workload is balanced with NN (symbolic memory percentage $\approx 20\%$). These findings highlight NSFlow's scalability to adapt to varying workloads and its efficiency in handling symbolic-heavy scenarios.

VII. CONCLUSION

To enable efficient NSAI for real-time cognitive applications, we propose NSFlow, the first end-to-end design automation framework to accelerate NSAI systems. NSFlow leverages the unique NSAI workload characteristics, explores dataflow and architecture design space, and generates scalable designs for FPGA deployment. We believe NSFlow paves the way for advancing efficient cognitive reasoning systems and unlocking new possibilities in NSAI.

REFERENCES

[1] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu, "The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision," *International Conference on Learning Representations (ICLR)*, 2019.

[2] C. Han, J. Mao, C. Gan, J. Tenenbaum, and J. Wu, "Visual concept-metaconcept learning," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[3] L. Mei, J. Mao, Z. Wang, C. Gan, and J. B. Tenenbaum, "Falcon: fast visual concept learning by integrating images, linguistic descriptions, and conceptual relations," *International Conference on Learning Representations (ICLR)*, 2022.

[4] K. Yi, C. Gan, Y. Li, P. Kohli, J. Wu, A. Torralba, and J. B. Tenenbaum, "Clevrer: Collision events for video representation and reasoning," in *International Conference on Learning Representations (ICLR)*, 2020.

[5] C. Zhang, B. Jia, S.-C. Zhu, and Y. Zhu, "Abstract spatial-temporal reasoning via probabilistic abduction and execution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9736–9746, 2021.

[6] V. Shah, A. Sharma, G. Shroff, L. Vig, T. Dash, and A. Srinivasan, "Knowledge-based analogical reasoning in neuro-symbolic latent spaces," *arXiv preprint arXiv:2209.08750*, 2022.

[7] T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong, "Solving olympiad geometry without human demonstrations," *Nature*, vol. 625, no. 7995, pp. 476–482, 2024.

[8] M. Ibrahim, Z. Wan, H. Li, P. Panda, T. Krishna, P. Kanerva, Y. Chen, and A. Raychowdhury, "Special session: Neuro-symbolic architecture meets large language models: A memory-centric perspective," in *2024 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pp. 11–20, IEEE, 2024.

[9] Z. Wan, C.-K. Liu, H. Yang, R. Raj, C. Li, H. You, Y. Fu, C. Wan, A. Samajdar, Y. C. Lin, *et al.*, "Towards cognitive ai systems: Workload and characterization of neuro-symbolic ai," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 268–279, IEEE, 2024.

[10] G. Booch, F. Fabiano, L. Horesh, K. Kate, J. Lenchner, N. Linck, A. Loreggia, K. Murgesan, N. Mattei, F. Rossi, *et al.*, "Thinking fast and slow in ai," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 15042–15046, 2021.

[11] G. Camposampiero, M. Hersche, A. Terzić, R. Wattenhofer, A. Sebastian, and A. Rahimi, "Towards learning abductive reasoning using vsa distributed representations," in *International Conference on Neural-Symbolic Learning and Reasoning*, pp. 370–385, Springer, 2024.

[12] M. Hersche, F. Di Stefano, T. Hofmann, A. Sebastian, and A. Rahimi, "Probabilistic abduction for visual abstract reasoning via learning rules in vector-symbolic architectures," *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[13] P. Hitzler, A. Eberhart, M. Ebrahimi, M. K. Sarker, and L. Zhou, "Neuro-symbolic approaches in artificial intelligence," *National Science Review*, vol. 9, no. 6, p. nwac035, 2022.

[14] K. Hamilton, A. Nayak, B. Božić, and L. Longo, "Is neuro-symbolic ai meeting its promises in natural language processing? a structured review," *Semantic Web*, vol. 15, no. 4, pp. 1265–1306, 2024.

[15] Z. Wan, C.-K. Liu, H. Yang, R. Raj, C. Li, H. You, Y. Fu, C. Wan, S. Li, Y. Kim, *et al.*, "Towards efficient neuro-symbolic ai: From workload characterization to hardware architecture," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2024.

[16] Z. Wan, C.-K. Liu, M. Ibrahim, H. Yang, S. Spetalnick, T. Krishna, and A. Raychowdhury, "H3dfact: Heterogeneous 3d integrated cim for factorization with holographic perceptual representations," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2024.

[17] M. Hersche, M. Zeqiri, L. Benini, A. Sebastian, and A. Rahimi, "A neuro-vector-symbolic architecture for solving raven's progressive matrices," *Nature Machine Intelligence*, vol. 5, no. 4, pp. 363–375, 2023.

[18] Z. Wan, C.-K. Liu, H. Yang, C. Li, H. You, Y. Fu, C. Wan, T. Krishna, Y. Lin, and A. Raychowdhury, "Towards cognitive ai systems: a survey and prospective on neuro-symbolic ai," *arXiv preprint arXiv:2401.01040*, 2024.

[19] J. Wang, L. Guo, and J. Cong, "Autosa: A polyhedral compiler for high-performance systolic arrays on fpga," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 93–104, 2021.

[20] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, pp. 1–6, 2017.

[21] Z. Li, Y. Zhang, J. Wang, and J. Lai, "A survey of fpga design for ai era," *Journal of Semiconductors*, vol. 41, no. 2, p. 021402, 2020.

[22] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "Understanding the potential of fpga-based spatial acceleration for large language model inference," *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

[23] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "A comprehensive evaluation of fpga-based spatial acceleration of llms," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 185–185, 2024.

[24] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang, *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 223–234, 2024.

[25] W. Huang, H. Wu, Q. Chen, C. Luo, S. Zeng, T. Li, and Y. Huang, "Fpga-based high-throughput cnn hardware accelerator with high computing resource utilization ratio," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 8, pp. 4069–4083, 2021.

[26] S. Yin, S. Tang, X. Lin, P. Ouyang, F. Tu, L. Liu, and S. Wei, "A high throughput acceleration for hybrid neural networks with efficient resource management on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 678–691, 2018.

[27] H. Xu, Y. Li, and S. Ji, "Llamaf: An efficient llama2 architecture accelerator on embedded fpgas," *arXiv preprint arXiv:2409.11424*, 2024.

[28] N. Menet, M. Hersche, G. Karunaratne, L. Benini, A. Sebastian, and A. Rahimi, "Mimonets: Multiple-input-multiple-output neural networks exploiting computation in superposition," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[29] Z. Wan, H. Yang, R. Raj, C.-K. Liu, A. Samajdar, A. Raychowdhury, and T. Krishna, "Cogsys: Efficient and scalable neurosymbolic cognition system via algorithm-hardware co-design," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 775–789, IEEE, 2025.

[30] M. Langhammer, S. Gribok, and G. Baeckler, "High density 8-bit multiplier systolic arrays for fpga," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 84–92, IEEE, 2020.

[31] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 58–68, IEEE, 2020.

[32] C. Zhang, F. Gao, B. Jia, Y. Zhu, and S.-C. Zhu, "Raven: A dataset for relational and analogical visual reasoning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 5317–5327, 2019.

[33] S. Hu, Y. Ma, X. Liu, Y. Wei, and S. Bai, "Stratified rule-aware network for abstract visual reasoning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 1567–1574, 2021.

[34] D. Barrett, F. Hill, A. Santoro, A. Morcos, and T. Lillicrap, "Measuring abstract reasoning in neural networks," in *International conference on machine learning (ICML)*, pp. 511–520, PMLR, 2018.

[35] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.

[36] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE micro*, vol. 40, no. 3, pp. 20–29, 2020.

[37] A. Zerroug, M. Vaishnav, J. Colin, S. Musslick, and T. Serre, "A benchmark for compositional visual reasoning," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 29776–29788, 2022.

[38] F. Fleuret, T. Li, C. Dubout, E. K. Wampler, S. Yantis, and D. Geman, "Comparing machines and humans on a visual categorization test," *Proceedings of the National Academy of Sciences*, vol. 108, no. 43, pp. 17621–17625, 2011.

Mixed-Precision Quantization for Deep Vision Models with Integer Quadratic Programming

Zihao Deng*
The University of Texas at Austin
zihaodeng@utexas.edu

Sayeh Sharify*
d-Matrix Corp.
sayehs@d-matrix.ai

Xin Wang
d-Matrix Corp.
xwang@d-matrix.ai

Michael Orshansky
The University of Texas at Austin
orshansky@utexas.edu

***Abstract*—Quantization is a widely used technique to compress neural networks. Assigning uniform bit-widths across all layers can result in significant accuracy degradation at low precision and inefficiency at high precision. Mixed-precision quantization (MPQ) addresses this by assigning varied bit-widths to layers, optimizing the accuracy-efficiency trade-off. Existing sensitivity-based methods for MPQ assume that quantization errors across layers are independent, which leads to suboptimal choices. We introduce CLADO, a practical sensitivity-based MPQ algorithm that captures cross-layer dependency of quantization error. CLADO approximates pairwise cross-layer errors using linear equations on a small data subset. Layerwise bit-widths are assigned by optimizing a new MPQ formulation based on cross-layer quantization errors using an Integer Quadratic Program. Experiments with CNN and vision transformer models on ImageNet demonstrate that CLADO achieves state-of-the-art mixed-precision quantization performance. Code repository available here.[1]**

I. Introduction

Reducing the storage and computational demands of deep neural networks (DNNs) is essential for practical applications. Model quantization effectively compresses DNNs [5, 6, 16, 22, 29] and, when applied to pretrained models, is known as post-training quantization (PTQ). PTQ typically requires only a small dataset to calibrate activation and weight statistics, allowing efficient determination of optimal quantization ranges for each layer. This calibration process is both data-efficient and quick to deploy, with bit-widths specified for each layer.

Uniform precision quantization (UPQ), the simplest quantization approach, assigns the same bit-width to all layers but overlooks layer-specific tolerance to quantization. However, research has shown that certain layers are more robust to quantization than others [2, 3, 10]; while quantizing some layers to low bit-width may yield minimal performance loss, even moderate quantization of other layers can lead to significant degradation. To take advantage of this phenomenon, mixed precision quantization (MPQ) optimizes bit-width assignments across layers such that lower prediction error is achieved at the same target compression rate as UPQ. Naïvely formulated, the search space of MPQ grows *exponentially* with the number of layers (L) to quantize. To address this, prior work presented two classes of solutions: *search-based methods* and *sensitivity-based methods* [27].

Search-based methods adopt search processes in which a large number of bit-width assignments are tested [1, 8, 12, 20, 28, 31]. These search-based methods are usually expensive, hard to parallelize, and will take hundreds or even thousands of GPU hours because of their iterative search nature. Sensitivity-based methods, on the other hand, evaluate a closed-form metric that measures the tolerance of layers to quantization [2, 3, 4, 9, 17, 21, 27, 32]. Such a metric, *viz. sensitivity*, is a property of each quantized layer. Various formulations have been proposed for the sensitivity metric in practice. Despite the diversity in the formulation of sensitivity, all sensitivity-based MPQ methods minimize total sum of sensitivities across layers, constrained by a target compression ratio. Such an optimization problem is much more efficient to solve. These methods take several minutes to a few GPU-hours to complete the entire MPQ optimization process. Importantly, their runtime can be further reduced by parallel computation of sensitivities and these sensitivities can be reused when requirements change. *Our work is also sensitivity-based.*

A key limitation of existing sensitivity-based MPQ algorithms is their assumption of independent layer-wise quantization effects, or dependence only within local blocks, as seen in BRECQ [18]. This assumption allows for an objective function that is simple to formulate and optimize but overlooks the interactions between quantized layers—what we refer to as the *cross-layer dependency of quantization error*. This oversight is especially impactful in computer vision models, where we demonstrate that an MPQ solution accounting for these dependencies outperforms methods assuming layer independence.

To overcome this limitation, we propose CLADO (**C**ross-**LA**yer-**D**ependency-aware **O**ptimization), an algorithm that measures the cross-layer dependency for all layers efficiently and optimizes a new cross-layer dependency aware objective via Integer Quadratic Programming (IQP). Following prior work, we start by applying the second-order Taylor expansion on network quantization loss. Then, we decompose the loss into two components with each being a sum of terms. Terms in the first component are specific to individual layers and they act independently across layers. They are well-addressed in prior work and we refer to them as the *layer-specific sensitivities*. Terms in the second component, which prior work ignores, represent interactions between layers and we refer to them as the *cross-layer sensitivities*. Naïvely evaluating them requires calculation of inter-layer Hessian matrices, which is practically infeasible. We propose an efficient backpropagation-free method that avoids Hessian calculation and computes sensitivities by solving a set of linear equations that requires $O(L^2)$ times evaluations of the network on a small set of samples. These sensitivities then allow us to reformulate the original MPQ problem as an IQP that contains $O(L)$ binary variables with linear constraints. The contributions of our work are:

- CLADO, the first sensitivity-based algorithm that captures the interactions of quantization error across *all layers* and transforms the MPQ problem into an *Integer Quadratic Program* that can be solved within seconds.
- CLADO is enabled by an efficient method to compute the cross-layer dependencies in $O(L^2)$ time. It is backpropagation-free and only requires forward evaluations of DNNs on a small amount of training data.
- Experiments on computer vision models, including ResNets, MobileNets, and ViT, show that CLADO achieves state-of-the-art mixed-precision performance.

*These authors contributed equally to this work.
[1] https://github.com/JamesTuna/CLADO_MPQ

II. Prior Work

Existing mixed-precision quantization techniques can be divided into two classes: search-based and sensitivity based.

Search-based methods: These methods evaluate the performance of mixed-precision quantized networks and use it to guide a search process. HAQ [28] and AutoQ [20] use Reinforcement Learning (RL), where all possible bit-width assignments of layers form the action space and the MPQ performance acts as the reward. Other search methods, such as MPQDNAS [31] and SPOS [12], apply Neural Architecture Search (NAS) to make MPQ a differentiable search process. The advantage of these methods is that they make decisions based on explicit evaluation of the quantized model's loss. However, they are computationally demanding, usually taking hundreds or even thousands of GPU-hours to complete [27]. Moreover, because each search is tied to a fixed set of constraints, when requirements change, new search processes need to be launched from scratch for search-based methods.

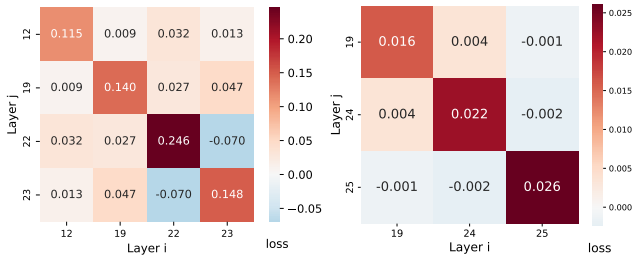
Sensitivity-based methods: These methods pre-compute *layer sensitivity* as a proxy metric (or "critic") to assess the impact of quantization on model performance. The sensitivities of layers in certain quantization precision are usually fast to measure, and once measured, they are reused in estimating the optimization objective as a function of different combinations of layer-wise precision assignments. This type of methods, including ZeroQ [2], variants of HAWQ [9, 10] and MPQCO [4], formulates the MPQ as a constrained combinatorial optimization problem. The objective to be minimized is the sum of layer sensitivities that quantify the overall quantization impact, while the constraints ensure certain target compression requirements, such as memory and/or computational budgets. Other works, such as OMPQ [21], BRECQ [18], and Bisection sensitivity-search [3], also fall into the category of sensitivity-based methods and propose different approaches for measuring layer sensitivity. For layer sensitivities defined in some of the above mentioned work, EMQ [8] studied the consistency between them and the actual network performance. We note that there is also prior work, BRECQ [18], that defines sensitivities on blocks of adjacent layers. BRECQ shares the same methodologies as the aforementioned approach in formulating the MPQ as an optimization problem with its objective being the summation of sensitivities and its constraints being the compression requirement.

To measure the layer sensitivities, a small subset of training data is needed (*e.g.*, 1024 training samples for ImageNet). We refer to it as the *sensitivity set*. Since sensitivity-based methods optimize a proxy, instead of the real increase in network loss, they do not require iterative model evaluation and are efficient to solve. State-of-the-art sensitivity-based algorithms achieve performance comparable to that of search-based algorithms [4]. *Our proposed method CLADO is also sensitivity-based.*

III. Cross-layer Dependency & MPQ Optimality

Empirically, cross-layer dependencies play a non-negligible role in the search of optimal MPQ solutions and ignoring them may lead to suboptimal solutions. Here we present, on ImageNet, two examples of MPQ suboptimality caused by ignoring the cross-layer dependencies. To make the suboptimality easy to understand, we focus on only a few layers and the goal is to *choose two layers to quantize so that the quantization loss is minimized*.

Figure 1a shows the sensitivity matrix of ResNet-34 under 2-bit quantization for four specific layers: `layer2.2.conv2` (index 12), `layer3.1.conv2` (index 19), `layer3.3.conv1` (index 22), and `layer3.3.conv2` (index 23). Ignoring the cross-layer interactions



(a) ResNet-34 (b) ResNet-50

Fig. 1: Sensitivity matrix of ResNet models. Entry (i, i): increase in loss caused by quantizing a single layer i. Entry (i, j): extra increase in loss when quantizing a pair of layers (i, j), compared to individually quantizing them.

(off-diagonal terms) leads to the selection of layers $(12, 19)$ because it gives the smallest change in predicted loss $0.115 + 0.140 = 0.255$. However, the real induced loss change of quantizing layers $(12, 19)$ is $0.115 + 0.140 + 2 \times 0.009 = 0.273$ and the optimal solution is to quantize layers $(22, 23)$ because it has the smallest induced change in loss $0.246 + 0.148 + 2 \times (-0.070) = 0.254$. Similarly, Figure 1b shows the sensitivity matrix of ResNet-50 under 4-bit quantization for three layers: `layer2.2.conv3` (index 19), `layer3.0.conv2` (index 24), and `layer3.0.conv3` (index 25). Ignoring cross-layer interactions would lead to selecting layers $(19, 24)$, as this combination yields the smallest predicted loss change, calculated as $0.016 + 0.022 = 0.038$. However, the actual induced loss change is $0.038 + 2 \times 0.004 = 0.046$. The optimal solution instead is to quantize layers $(19, 25)$, yielding the smallest induced loss change of $0.016 + 0.026 + 2 \times (-0.001) = 0.040$.

IV. CLADO: Cross Layer Dependency aware Optimization

A. Preliminaries

We first introduce the formulation of the MPQ problem following Chen et al. [4].

Notation: We assume an I-layer (indices start from 0) neural network $f : \mathbf{\Theta} \times \mathbb{X} \to \mathbb{Y}$ and a training dataset of N samples $\left(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\right) \in \mathbb{X} \times \mathbb{Y}$ with $n \in \{1, \ldots, N\}$. The model maps each sample $\mathbf{x}^{(n)}$ to a prediction $\hat{\mathbf{y}}^{(n)} = f\left(\theta, \mathbf{x}^{(n)}\right)$, using some parameters $\theta \in \mathbf{\Theta}$. Then the predictions are compared with the ground truth $\mathbf{y}^{(n)}$ and evaluated with a task-specific loss function $\ell : \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}$. This leads to the objective function to minimize $\mathcal{L} : \mathbf{\Theta} \to \mathbb{R}$:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \ell\left(f\left(\theta, \mathbf{x}^{(n)}\right), \mathbf{y}^{(n)}\right) = \frac{1}{N} \sum_{n=1}^{N} \ell^{(n)}(\theta) \tag{1}$$

We denote the weight tensor of the i^{th} layer as $W^{(i)} \in \mathbb{R}^{c_o \times c_{in} \times k \times k}$ and its flattened version as $w^{(i)} \in \mathbb{R}^{c_o c_{in} k^2}$, where k is the kernel size for a convolutional layer and $k = 1$ for fully-connected layers, c_{in} and c_o are the number of input and output channels, respectively. The quantization function is denoted by $Q : \mathbb{R}^D \times \mathbb{Z}^+ \to \Pi_b$, which takes a full-precision vector and the quantization bit-width as input and produces the quantized vector. Unless otherwise stated, we used per-tensor uniform symmetric quantization as it is the simplest scheme in practice due to its ease of implementation in hardware. As a result, $\Pi_b = s \times \left\{-2^{b-1}, \ldots, 0, \ldots, 2^{b-1} - 1\right\}$ for signed input and $s \times \left\{0, \ldots, 2^b - 1\right\}$ for unsigned one, where b is the quantization bit-width and s is the quantization scale factor. The quantized b-bit

weight is given by $Q(w,b) = \text{clip}\left(\lfloor w/s \rceil, -2^{b-1}, 2^{b-1}-1\right) \times s$ for full precision w.

Discrete constrained optimization formulation: Let $w \triangleq \left\{w^{(i)}\right\}_{i=0}^{I-1}$ be the set of flattened weights of the network with I layers. To find the optimal bit-width assignment with the goal of minimizing total model size, MPQ can be written as the following discrete constrained problem. $\mathbb{B} = \{b_1, b_2, ..., b_{|\mathbb{B}|}\}$ denotes the set of bit-widths to be assigned in MPQ. In this work, C_{target} is the target model size of the network, and $|\cdot|$ denotes the length of vectors.

$$\begin{aligned} \min_{\{b^{(i)}\}} \ & \frac{1}{N}\sum_{n=1}^{N} \ell\left(f\left(w+\Delta w, \mathbf{x^{(n)}}\right), \mathbf{y^{(n)}}\right) \\ \text{s.t.} \quad & \Delta w^{(i)} = Q\left(w^{(i)}, b^{(i)}\right) - w^{(i)} \\ & \sum_i \left|w^{(i)}\right| \cdot b^{(i)} \le C_{\text{target}} \\ & b^{(i)} \in \mathbb{B}, i \in \{0, \dots, I-1\} \end{aligned} \tag{2}$$

Proxy of optimization objective: Let $g_w \triangleq \nabla \mathcal{L}(w)$ and $H_w \triangleq \nabla^2 \mathcal{L}(w)$ be the gradient and the Hessian, respectively. For computational tractability, CLADO approximates the objective via the second-order Taylor expansion:

$$\begin{aligned} \mathcal{L}(w+\Delta w) &= \frac{1}{N}\sum_{n=1}^{N} \ell^{(n)}(w+\Delta w) \\ &\approx \mathcal{L}(w) + g_w^T \Delta w + \frac{1}{2}\Delta w^T H_w \Delta w \end{aligned} \tag{3}$$

For a well-trained model, it is reasonable to assume that training has converged to a local minimum, and thus, the gradient $g_w \approx 0$, resulting in:

$$\mathcal{L}(w+\Delta w) \approx \mathcal{L}(w) + \frac{1}{2}\Delta w^T H_w \Delta w \tag{4}$$

In essence, MPQ seeks to minimize $\mathcal{L}(w+\Delta w)$ as a function of Δw, treating $\mathcal{L}(w)$ as a constant; thus, minimizing $\mathcal{L}(w+\Delta w)$ is equivalent to minimizing $\frac{1}{2}\Delta w^T H_w \Delta w$. Here we use $\Delta w^T H_w \Delta w$ as a proxy of the objective in (2).

B. Methods

The core of CLADO is a way to capture in full the pairwise interactions of quantization error between all layers, which is either ignored, or only partially addressed in prior work.

Layer-specific and cross-layer sensitivities: Let H_{ij}s be the partition of the Hessian at layer-level granularity. Specifically, let $H_{ii} \triangleq \partial^2 \mathcal{L}(w)/\partial {w^{(i)}}^2$ be the Hessian of layer i and $H_{ij} \triangleq \partial^2 \mathcal{L}(w)/\partial w^{(i)} \partial w^{(j)}$ be the cross-layer Hessian between layers i and j. Unless otherwise stated, we use i, j to denote the indices of layers and they take values from $\{0, 1, .., I-1\}$, where I is the number of layers in the network. Then,

$$\Delta w^T H_w \Delta w = \sum_i \Delta w^{(i)T} H_{ii} \Delta w^{(i)} + \sum_{i \neq j} \Delta w^{(i)T} H_{ij} \Delta w^{(j)} \tag{5}$$

Equation 5 decomposes the objective into two terms. The first term captures the contribution to loss increase from individual layers due to the intra-layer effects resulted from quantization. We refer to them as *layer-specific sensitivities*. The second part contains the contributions from effects of pairwise layer interaction as a consequence of two different layers being jointly quantized. We refer to them as the *cross-layer sensitivities*. We use $\Omega_{i,j}(\cdot,\cdot)$ as a unified notation for both types of sensitivities.

$$\Omega_{i,j}(\Delta w^{(i)}, \Delta w^{(j)}) \triangleq \Delta w^{(i)T} H_{ij} \Delta w^{(j)} \tag{6}$$

We rewrite (5) in terms of Ω as follows; minimizing (5) is equivalent to minimizing:

$$\Omega = \sum_i \Omega_{i,i}(\Delta w^{(i)}, \Delta w^{(i)}) + \sum_{i \neq j} \Omega_{i,j}(\Delta w^{(i)}, \Delta w^{(j)}) \tag{7}$$

Note that this re-definition obviates direct reference to the Hessian. We will show below that $\Omega_{i,j}$ can be evaluated without computing the Hessian. Unlike prior work [2, 4, 9, 10] that ignores the cross-layer interaction terms, here we optimize (7) in its full form:

$$\begin{aligned} \min_{\{b^{(i)}\}} \ & \Omega \\ \text{s.t.} \quad & \Delta w^{(i)} = Q\left(w^{(i)}, b^{(i)}\right) - w^{(i)} \\ & \sum_i \left|w^{(i)}\right| \cdot b^{(i)} \le C_{\text{target}} \\ & b^{(i)} \in \mathbb{B}, i \in \{0, \dots, I-1\} \end{aligned} \tag{8}$$

IQP formulation: Next we show that (8) can be formulated as an Integer Quadratic Program (IQP) that solves for the bit-width assignment decisions. Since each layer i has $|\mathbb{B}|$ candidate bit-width choices:$\{b_1, b_2, .., b_{|\mathbb{B}|}\}$, $\Delta w^{(i)}$ can take $|\mathbb{B}|$ values: $\Delta w^{(i)} \in \{\Delta w_1^{(i)}, \Delta w_2^{(i)}, .., \Delta w_{|\mathbb{B}|}^{(i)}\}$. Here, we use $\Delta w_m^{(i)} \triangleq Q(w^{(i)}, b_m) - w^{(i)}$ with $m \in \{1, 2, .., |\mathbb{B}|\}$ to denote the quantization error on $w^{(i)}$, when quantizing it to b_m bits. For each layer i, we introduce a one-hot variable $\alpha^{(i)} \in \{0,1\}^{|\mathbb{B}|}$s.t. $\sum_{m=1}^{|\mathbb{B}|} \alpha_m^{(i)} = 1$ to represent this layer's bit-width decision. The single entry of 1 in $\alpha^{(i)}$, *e.g.* $\alpha_m^{(i)}$, indicates the selection of a corresponding $\Delta w_m^{(i)}$ and the chosen bit-width b_m. For compact notation, let $\alpha \in \mathbb{R}^{|\mathbb{B}|I}$ be the concatenation of $\alpha^{(i)}$s, i.e., $\alpha = (\alpha^{(0)}, \alpha^{(1)}, ..\alpha^{(I-1)})$. Specifically,

$$\begin{aligned} & \Delta w^{(i)} = \alpha_1^{(i)} \Delta w_1^{(i)} + \alpha_2^{(i)} \Delta w_2^{(i)} + ... + \alpha_{|\mathbb{B}|}^{(i)} \Delta w_{|\mathbb{B}|}^{(i)} \\ & \Delta w_m^{(i)} = Q(w^{(i)}, b_m) - w^{(i)},\ m \in \{1, 2, ..., |\mathbb{B}|\} \\ & \alpha_1^{(i)}, \alpha_2^{(i)}, ..., \alpha_{|\mathbb{B}|}^{(i)} \in \{0,1\}, \alpha_1^{(i)} + \alpha_2^{(i)} + ... + \alpha_{|\mathbb{B}|}^{(i)} = 1 \end{aligned} \tag{9}$$

In addition, we gather all $\Omega_{i,j}$s into a matrix $\hat{G} \in \mathbb{R}^{|\mathbb{B}|I \times |\mathbb{B}|I}$, to which we refer to as the *sensitivity matrix*. Specifically, for $m, n \in \{1, 2, .., |\mathbb{B}|\}$:

$$\hat{G}_{|\mathbb{B}|i+m, |\mathbb{B}|j+n} \triangleq \Omega_{i,j}(\Delta w_m^{(i)}, \Delta w_n^{(j)}) \tag{10}$$

Expanding the $\Delta w^{(i)}, \Delta w^{(j)}$ terms in (5) by (9) and bringing in the definition of $\hat{G}$ in (10), we have $\Omega = \Delta w^T H_w \Delta w = \alpha^T \hat{G} \alpha$. Accordingly, the MPQ problem is equivalent to the following IQP:

$$\begin{aligned} & \min_{\{\alpha^{(i)}\}} \alpha^T \hat{G} \alpha \\ & \text{s.t. } \alpha = (\alpha^{(1)}, \alpha^{(2)}, ..\alpha^{(I)}) \\ & \alpha_1^{(i)}, \alpha_2^{(i)}, ..., \alpha_{|\mathbb{B}|}^{(i)} \in \{0,1\}, \alpha_1^{(i)} + \alpha_2^{(i)} + ... + \alpha_{|\mathbb{B}|}^{(i)} = 1 \\ & \textstyle\sum_i \sum_m \alpha_m^{(i)} \left|w^{(i)}\right| \cdot b_m \le C_{\text{target}} \\ & i \in \{0, \dots, I-1\},\ m \in \{1, \dots, |\mathbb{B}|\} \end{aligned} \tag{11}$$

Backpropagation-free sensitivity measurement: We now present an efficient way to compute $\hat{G}$ through only forward evaluations of DNNs. For any pair of layers i, j and their corresponding bit-width choices m, n, computing $\Omega_i(\Delta w_m^{(i)})$, $\Omega_j(\Delta w_n^{(j)})$, and $\Omega_{i,j}(\Delta w_m^{(i)}, \Delta w_n^{(j)})$ in a naïve manner requires computing the Hessians. However, we point out that the Hessian computation can be avoided. We estimate Ω by directly measuring the change in loss due to network perturbations:

$$\begin{aligned} & \Omega_{i,i}(\Delta w_m^{(i)}, \Delta w_m^{(i)}) \approx 2\left(\mathcal{L}(w + \Delta w_m^{(i)}) - \mathcal{L}(w)\right) \\ & \Omega_{j,j}(\Delta w_n^{(j)}, \Delta w_n^{(j)}) \approx 2\left(\mathcal{L}(w + \Delta w_n^{(j)}) - \mathcal{L}(w)\right) \\ & \Omega_{i,i}(\Delta w_m^{(i)}, \Delta w_m^{(i)}) + \Omega_{j,j}(\Delta w_n^{(j)}, \Delta w_n^{(j)}) + 2\Omega_{i,j}(\Delta w_m^{(i)}, \Delta w_n^{(j)}) \\ & \approx 2\left(\mathcal{L}(w + \Delta w_m^{(i)} + \Delta w_n^{(j)}) - \mathcal{L}(w)\right) \end{aligned} \tag{12}$$

Equations in (12) follow from equation (4). The right sides of the three equations in (12) are computed by taking the difference in the loss of the quantized model from that of the full-precision model. Subtracting the first two equations from the third equation:

$$\begin{aligned} \Omega_{i,j}(\Delta w_m^{(i)}, \Delta w_n^{(j)}) \approx \ & \mathcal{L}(w + \Delta w_m^{(i)} + \Delta w_n^{(j)}) + \mathcal{L}(w) \\ & - \mathcal{L}(w + \Delta w_m^{(i)}) - \mathcal{L}(w + \Delta w_n^{(j)}) \end{aligned} \tag{13}$$

Algorithm 1 CLADO

Input: I-layer network with bit-width options $\mathbb{B} = \{b_1, b_2, .., b_{|\mathbb{B}|}\}$ and model size target C_{target}.
Output: The bit-width decision for each layer.
Evaluate the loss of full-precision model: $\mathcal{L}(w)$
Initialize $\hat{G} \in \mathbb{R}^{|\mathbb{B}|I \times |\mathbb{B}|I}$
for i in $\{0, \ldots, I-1\}$ **do**
  for m in $\{1, \ldots, |\mathbb{B}|\}$ **do**
    $\Delta w^{(i)} \leftarrow Q(w^{(i)}, b_m) - w^{(i)}$ // Quantize layer i to b_m bits
    Evaluate loss of quantized network $\mathcal{L}(w + \Delta w^{(i)})$
    $S \leftarrow 2(\mathcal{L}(w + \Delta w^{(i)}) - \mathcal{L}(w))$ // layer-specific sensitivity
    $\hat{G}_{|\mathbb{B}|i+m, |\mathbb{B}|i+m} \leftarrow S$ // Store layer-specific sensitivity
  end for
end for
for i in $\{0, \ldots, I-2\}$ **do**
  for j in $\{i+1, \ldots, I-1\}$ **do**
    for (m, n) in $\{1, ..|\mathbb{B}|\} \times \{1, ..|\mathbb{B}|\}$ **do**
      // Quantize layer i to b_m bits and layer j to b_n bits
      $\Delta w^{(i)} \leftarrow Q(w^{(i)}, b_m) - w^{(i)}$
      $\Delta w^{(j)} \leftarrow Q(w^{(j)}, b_n) - w^{(j)}$
      Measure loss of quantized model $\mathcal{L}(w + \Delta w^{(i)} + \Delta w^{(j)})$
      // Compute cross-layer sensitivities using equation (13)
      $S \leftarrow \mathcal{L}(w + \Delta w^{(i)} + \Delta w^{(j)}) + \mathcal{L}(w)$
      $S \leftarrow S - 0.5\hat{G}_{|\mathbb{B}|i+m, |\mathbb{B}|i+m}$
      $S \leftarrow S - 0.5\hat{G}_{|\mathbb{B}|j+n, |\mathbb{B}|j+n}$
      $\hat{G}_{|\mathbb{B}|i+m, |\mathbb{B}|j+n} \leftarrow S$
    end for
  end for
end for
Compute the eigen-decomposition: $\hat{G} = \sum_{i=1}^{|\mathbb{B}|I} u_i e_i u_i^T$ and set PSD approximated $\hat{G} \leftarrow \sum_{i=1, e_i>0}^{|\mathbb{B}|I} u_i e_i u_i^T$
Solve the IQP problem in (11) by plugging in $\hat{G}$,$\mathbb{B}$, and C_{tagret} to get the optimal solution α^*.
Return: The bit-width decisions indicated by α^*.

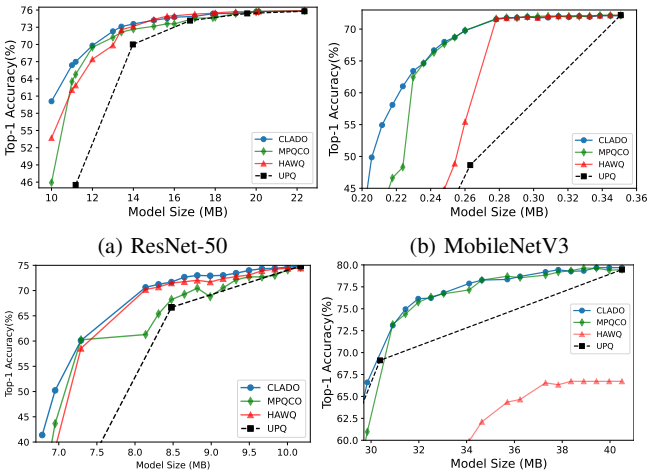
Importantly, (12) and (13) compute all sensitivities by *only forward evaluations* of DNNs on the small sensitivity set. In total, CLADO sensitivity computation requires $\frac{1}{2}|\mathbb{B}|I(|\mathbb{B}|I+1)$ measurements of the network on a small set of samples. After sensitivity computation, we apply a positive semi-definite (PSD) approximation to the matrix $\hat{G}$. Ideally, G is PSD if computed on the entire training set after full convergence, but using a small sensitivity set can introduce measurement errors, making it indefinite. In practice, PSD approximation is crucial for meaningful MPQ solutions (see Section VII for details).

V. Experimental Results: CNNs

A. Experimental Settings

Dataset and models: We conduct experiments on the ImageNet dataset [26]. To compare CLADO to existing work, we test two state-of-the-art algorithms: MPQCO [4] and HAWQ-V3 [32]. We evaluate with four convolutional models: ResNet-34/50 [14], MobileNetV3 [15], and RegNet-3.2GF [25]. For all models excluding MobileNetV3, we quantize activations to 8-bit and consider three candidate bit-widths for weights: $\mathbb{B} = \{2, 4, 8\}$. (Note that, as shown in Algorithm 1, CLADO can work with any number ($|\mathbb{B}|$) of bit-width candidates, for experiments, we use $|\mathbb{B}| = 3$.) We consider more conservative quantization bit-width candidates $\mathbb{B} = \{4, 6, 8\}$ for MobileNetV3 because of its high parameter efficiency.

Implementation details: All algorithms are implemented in PyTorch [23]. The publicly available package CVXPY [7] is used to solve the IQP problem in CLADO with GUROBI [13] as its backend. For fair comparisons, all algorithms are run using the same procedures and settings. We first download pretrained full-precision models from TorchVision. Then, model quantization is performed using MQBench [19]. Following MPQCO, quantization scale factors (and zero points in the affine case) are determined by minimization of the MSE between the `float32` values and their quantized values. After MQBench quantization, different MPQ algorithms are tested for mixed-precision quantization.



(a) ResNet-50 (b) MobileNetV3

(c) RegNet-3.2GF (d) ViT-base

Fig. 2: CNNs and ViT on the ImageNet dataset

Use of multiple sensitivity sets: Results of sensitivity-based algorithms depend on the samples in the sensitivity set and show some variation. To study the dependence of algorithms on the sensitivity set and to reduce the variation in results, we use multiple sensitivity sets: for each sample size ranging from 256 to 4096, we randomly construct 24 sets with the given size. Then, we test the performance of algorithms on each of the 24 sets.

B. Comparison of Different MPQ Algorithms

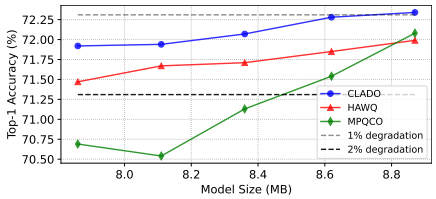
Runtime: CLADO requires $\frac{1}{2}|\mathbb{B}|I(|\mathbb{B}|I+1)$ measurements of the network output on the small sensitivity set. On a single Nvidia RTX2080 GPU, our implementation takes 1 hour to generate the sensitivities for the ResNet-34 and 2.5 hours for the ResNet-50 model. HAWQ takes roughly the same number of GPU-hours as CLADO to compute sensitivities. MPQCO is the fastest one, taking 5-10 minutes for its sensitivity estimation.

Average performance: Table I presents the averaged post-training quantization performance of algorithms under different model sizes (see Figure 2 for tradeoff curves with more data points.) CLADO delivers the best PTQ models under most constraints. Its advantage is especially prominent when models are aggressively compressed to low precisions. For example, under the smallest model size constraints in the table, CLADO demonstrates 5.73% (ResNet-34) to 32% (MobileNetV3) accuracy improvement over HAWQ and MPQCO. However, we note that the PTQ degradation in these low precision regions is too high and renders the results less meaningful. To address this, we apply additional quantization-aware fine-tuning. The QAT results are in Figure 3.

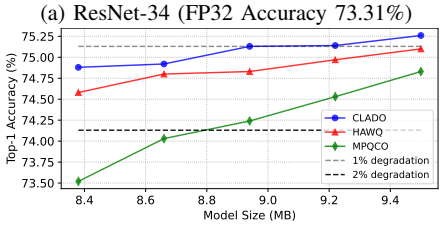
We found that, though CLADO is derived with (13) under the PTQ settings, its bit-width assignments remain superior to others after fine-tuning. Since QAT significantly reduces degradation, the difference between algorithms' after-QAT performance is much smaller. E.g., for 8.87MB ResNet-34, CLADO, HAWQ, MPQCO achieve pre-QAT accuracies of 44.40%, 9.96%, and 0.51%, respectively, and post-QAT accuracies of 72.34%, 72.00%, and 72.08%. *Although the after-QAT difference is small, CLADO maintains meaningful advantage: e.g.*, for ResNet-34 with 8.87MB size constraint and ResNet-50 with 8.94MB,

TABLE I: MPQ results (PTQ): CNNs and ViT on ImageNet. Numbers denote the averaged Top-1 accuracy (percentage) for different algorithms. CLADO delivers the best MPQ results under most model sizes. (*: for ablation study, CLADO with cross-layer terms removed. $^+$: experiment using per-channel affine instead of per-tensor symmetric quantization.)

Model	ResNet-34 INT8 size: 20.27MB; Acc: 73.10			ResNet-50 INT8 size: 22.36MB; Acc: 75.87			MobileNetV3-Large$^+$ INT8 size: 0.35MB; Acc: 72.20			RegNet-3.2GF INT8 size: 13.57MB; Acc: 77.88			ViT-base$^+$ INT4 size: 40.5MB; Acc: 79.47		
Size (MB)	10.13	12.16	14.19	10.00	13.42	17.89	0.21	0.25	0.29	6.78	9.50	12.89	29.84	31.97	34.11
HAWQ	56.86	68.13	72.00	53.70	72.55	75.36	15.15	44.88	**71.82**	33.84	73.08	77.24	10.57	52.44	59.91
MPQCO	53.39	67.51	71.53	45.90	72.13	74.64	17.64	67.62	71.75	29.34	72.73	77.51	60.96	75.73	77.15
CLADO*	54.47	70.37	71.67	53.22	72.51	75.35	43.80	67.52	71.75	32.84	73.48	77.51	54.42	70.75	74.05
CLADO	**62.59**	**70.38**	**72.20**	**60.13**	**73.10**	**75.42**	**49.85**	**68.00**	71.81	**41.40**	**73.98**	**77.53**	**66.60**	**76.15**	**77.88**



(a) ResNet-34 (FP32 Accuracy 73.31%)



(b) ResNet-50 (FP32 Accuracy 76.13%)

Fig. 3: MPQ results (QAT): QAT fine-tuning based on CLADO outperforms fine-tuning based on other MPQ methods. (Range of model sizes is chosen to be close to 3-bit UPQ. With higher model size, all MPQ algorithms tend to recover FP32 performance.)

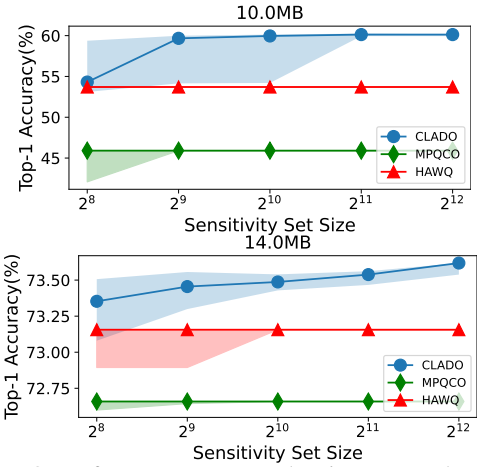


Fig. 4: MPQ performance vs. sample size. Data shows median performance across 24 random sensitivity sets. Colored regions cover the upper and lower performance quartiles.

9.22MB, and 9.50MB constraints, CLADO maintains $\leq 1\%$ accuracy degradation while others exhibit a higher degradation.

Dependence on the sensitivity set: Figure 4 shows algorithms' variation due to different samples of the sensitivity set. For each sample size, it presents, the median, the upper quartile (75%), and the lower quartile (25%) of algorithms' performance on the 24 randomly sampled sensitivity sets. CLADO produces reasonably stable results. Though it may have slightly higher variation sometimes, its lower quartile performance is almost always better than alternatives' upper quartile performance, which is especially true when the sample size is ≥ 1024 on ImageNet.

Analysis of bit-width assignments: Figure 5 visualizes the MPQ decisions using different algorithms on ResNet-50. Different MPQ algorithms share some commonalities, *e.g.*, generally they assign more bits to shallow layers and less bits to deep layers. This aligns with our observation that shallow layers have higher sensitivities and fewer parameters. However, algorithms decide quite differently for certain layers: compared to HAWQ and MPQCO, CLADO chooses to quantize more aggressively on the first convolutional layer of `layer1.1` (index 4), and `layer4.1` (index 46) and more conservatively on the downsampling layer of `layer3.0` (index 26).

VI. Experimental Results: Vision Transformer

We also tested our MPQ flow on a transformer-based vision model. The Vision Transformer (ViT) has significantly advanced computer vision by applying transformer architectures, originally designed for natural language processing, to visual data [11]. Its ability to capture long-range dependencies and global context has shown superior performance compared to traditional convolutional neural networks (CNNs) in tasks like image classification and object detection. ViT is also commonly used in CLIP [24] as an image encoder to extract visual features, enabling zero-shot image classification and connecting vision to text. We use a pretrained ViT-based model from the HuggingFace library [30] and evaluate the quantized model on the ImageNet dataset using various PTQ algorithms [26].

Table I shows the size/performance tradeoffs (see Figure 2d for full Pareto fronts). We found that *CLADO almost always produces the best MPQ solutions.* The advantage of CLADO over the next best alternative is small under large size constraints. Under more restrictive size constraints, CLADO becomes more advantageous.

VII. Ablation Studies

Cross-Layer dependencies: To demonstrate the importance of cross-layer dependencies, we test the performance of CLADO assuming zero cross-layer sensitivities for all layers. We refer to this set of experiments as "CLADO*". Compared to full CLADO, *ignoring cross-layer dependencies consistently and significantly degrades MPQ performance* (Table I). Cross-layer dependencies seem to be rather prominent for ViT and for CNN compression under small size constraints.

BRECQ [18] claimed that block-level optimization performs best for quantization by adaptive rounding. Interestingly, however, we show that this observation does not extend to MPQ. Following BRECQ, we conducted ablation experiments in which cross-layer dependencies are only considered within blocks of layers. In this set of experiments, we intentionally leave out the inter-block dependencies. We refer to this group of experiments as "block interactions" (the black curves in Figure 6). *We found that ignoring inter-block de-*

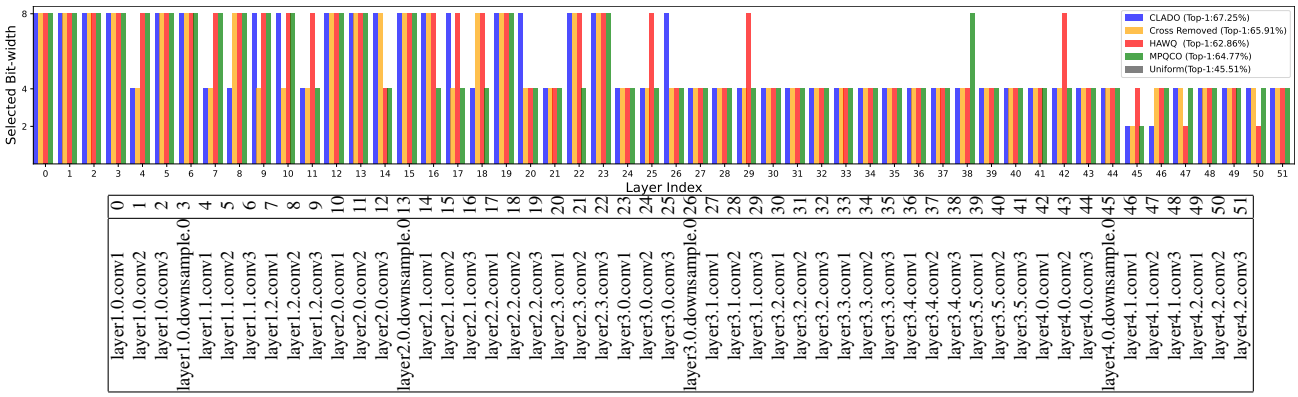
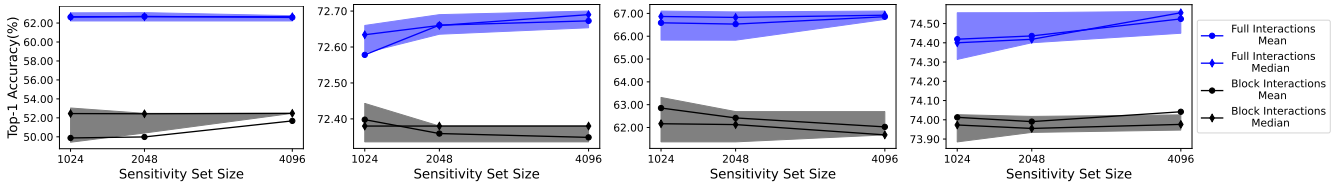
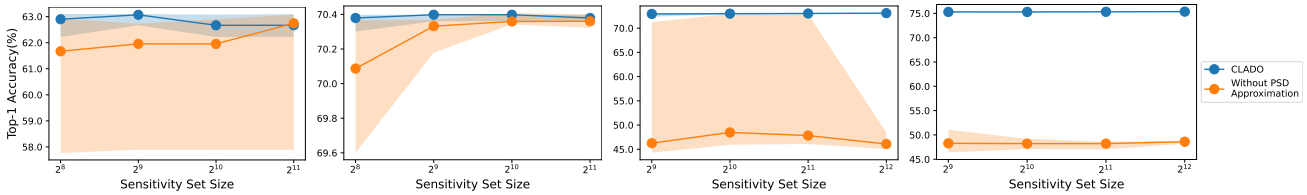


Fig. 5: Bit-width assignments to ResNet-50 along with the layer index-layer name mappings. Model size constraint is set to be 11.18MB, corresponding to 4-bit UPQ.



(a) ResNet-34: 10.14MB (b) ResNet-34: 16.22MB (c) ResNet-50: 11.18MB (d) ResNet-50: 15.65MB

Fig. 6: Leaving out inter-block dependencies worsens MPQ. Blue: CLADO, captures all-layer dependencies. Black: inter-block dependencies ignored, following BRECQ [18] Data represents median performance on 24 randomly sampled sensitivity sets. Colored regions cover the upper and lower quartiles of performance.



(a) ResNet-34, 10.1MB (b) ResNet-34, 12.2MB (c) ResNet-50, 13.4MB (d) ResNet-50, 17.9MB

Fig. 7: Ablation on PSD Approximation of the Sensitivity Matrix $\hat{G}$.

TABLE II: Approx. accuracy of the second-order quantization error (vHv): our fast method vs. the exact Hessian evaluation.

ResNet20 Layers	module.layer1 .0.conv1 (2-bit)	module.layer2 .1.conv2 (6-bit)	module.layer3 .2.conv1 (2-bit)	module.layer3 .2.conv1 (4-bit)	module.layer3 .2.conv1 (6-bit)	module.fc (2-bit)	module.fc (4-bit)
vHv (exact)	0.17105	0.00005	0.09384	0.00374	0.00037	0.03154	0.00057
vHv (ours)	0.14670	0.00006	0.09912	0.00362	0.00038	0.03190	0.00057

pendencies significantly worsens the MPQ solution. We hypothesize that MPQ, with far fewer parameters ($|\mathbb{B}|I$) than adaptive rounding, is less prone to the overfitting issue noted by BRECQ. Thus, using full pairwise interactions yields more robust results, while limiting to intra-block interactions likely underfits MPQ solutions.

PSD sensitivity matrix approximation: PSD approximation is critical to CLADO's superior performance over other studied algorithms. To explore this, we disable the use of PSD approximation in CLADO, keeping the rest of CLADO unchanged. We found that it makes CVXPY unable to converge on a solution in more than 3 hours (with the PSD approximation, the solver computes the optimal solutions within seconds). This observation aligns with our expectation that the convexity of the objective Ω makes the IQP more well-behaved and easier to solve. The results are captured by the purple curves in Figures 7: though PSD does not always produce best results, it improves their consistency, and, in some cases, prevents a severe degradation in the quality of solutions.

Approximation accuracy of our fast Hessian-free method: We evaluate the accuracy of our fast method that avoids the Hessian computation by comparing the quantization-induced loss changes with those from the exact Hessian method, which is 7× slower and requires more CUDA memory. Results for randomly selected shallow and deep layers in ResNet20 are shown in Table II, where v represents the quantization error at each bit-width. The approximation error is generally small.

VIII. Conclusions

We present CLADO, a sensitivity-based mixed-precision quantization algorithm that exploits cross-layer quantization error dependencies. We derive a sensitivity objective from 2^{nd}-order Taylor expansion of loss, and reformulate the bit-width assignment problem as an integer quadratic program that can be solved efficiently. We devise an efficient algorithm to compute the cross-layer dependencies that requires only forward evaluations of networks. Extensive experiments are conducted to demonstrate the effectiveness of the proposed method.

References

[1] S. Azizi, M. Nazemi, A. Fayyazi, and M. Pedram. Automated optimization of deep neural networks: Dynamic bit-width and layer-width selection via cluster-based parzen estimation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024.

[2] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer. ZeroQ: A novel zero shot quantization framework. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13166–13175, 2020.

[3] A. Chauhan, U. Tiwari, and V. N. R. Post training mixed precision quantization of neural networks using first-order information. *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 1335–1344, 2023.

[4] W. Chen, P. Wang, and J. Cheng. Towards mixed-precision quantization of neural networks via constrained optimization. *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5330–5339, 2021.

[5] X. Chu, L. Li, and B. Zhang. Make repvgg greater again: A quantization-aware approach. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:11624–11632, 2024.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[7] S. Diamond and S. P. Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of machine learning research : JMLR*, 17, 2016.

[8] P. Dong, L. Li, Z. Wei, X.-Y. Niu, Z. Tian, and H. Pan. EMQ: Evolving training-free proxies for automated mixed precision quantization. *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17030–17040, 2023.

[9] Z. Dong, Z. Yao, Y. Cai, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. *ArXiv*, abs/1911.03852, 2020.

[10] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 293–302, 2019.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.

[12] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020.

[13] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[15] A. G. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for mobilenetv3. *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

[16] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer. Integer-only bert quantization. *ArXiv*, abs/2101.01321, 2021.

[17] S. Kundu, S. Wang, Q. Sun, P. A. Beerel, and M. Pedram. BMPQ: bit-gradient sensitivity-driven mixed-precision quantization of dnns from scratch. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 588–591, 2022.

[18] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu. BRECQ: Pushing the limit of post-training quantization by block reconstruction. *ArXiv*, abs/2102.05426, 2021.

[19] Y. Li*, M. Shen*, J. Ma*, Y. Ren*, M. Zhao*, Q. Zhang*, R. Gong*, F. Yu, and J. Yan. Mqbench: Towards reproducible and deployable model quantization benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[20] Q. Lou, F. Guo, L. Liu, M. Kim, and L. Jiang. AutoQ: Automated kernel-wise neural network quantization. *ArXiv*, abs/1902.05690, 2020.

[21] Y. Ma, T. Jin, X. Zheng, Y. Wang, H. Li, G. Jiang, W. Zhang, and R. Ji. OMPQ: Orthogonal mixed precision quantization. *ArXiv*, abs/2109.07865, 2021.

[22] M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos, and T. Blankevoort. Up or down? adaptive rounding for post-training quantization. *ArXiv*, abs/2004.10568, 2020.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[24] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[25] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár. Designing network design spaces. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10425–10433, 2020.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2014.

[27] C. Tang, K. Ouyang, Z. Wang, Y. Zhu, Y. Wang, W. Ji, and W. Zhu. Mixed-precision neural network quantization via learned layer-wise importance. *ArXiv*, abs/2203.08368, 2022.

[28] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. HAQ: Hardware-aware automated quantization with mixed precision. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8612, 2019.

[29] X. Wei, R. Gong, Y. Li, X. Liu, and F. Yu. QDrop: Randomly dropping quantization for extremely low-bit post-training quantization. *ArXiv*, abs/2203.05740, 2022.

[30] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[31] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *ArXiv*, abs/1812.00090, 2018.

[32] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. Mahoney, et al. HAWQ-V3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR, 2021.

Optimizing windowed arithmetic for quantum attacks against RSA-2048

Alessandro Luongo*†, Varun Narasimhachar‡, Adithya Sireesh§
*Center for Quantum Technologies (CQT), Singapore
†Inveriant Pte. Ltd., Singapore
‡Agency for Science, Technology and Research (A*STAR), Singapore
§Quantum Software Lab, University of Edinburgh, Edinburgh, United Kingdom

***Abstract*—Windowed arithmetic is a technique for reducing the cost of quantum arithmetic circuits with space–time trade-offs using memory queries to precomputed tables. It can reduce the asymptotic cost of modular exponentiation from $\mathcal{O}(n^2)$ to $\mathcal{O}(n^2/\log^2 n)$ operations, resulting in the current state-of-the-art compilations of quantum attacks against modern cryptography. We introduce several optimizations to windowed arithmetic. Notably, we effect an approximate 50% reduction in the costs of uncomputing memory lookups in quantum factoring applications. We validate our optimizations by improving the gate count of quantum attacks against public-key cryptography by 1.5% to 3.4%, depending on the key size. We also enable a 16% runtime reduction at the cost of a 12% increase in qubit count. Our techniques can be used to reduce the complexity of not only factoring algorithms but also a wide range of quantum algorithms that rely on windowed arithmetic.**

***Index Terms*—modular arithmetic, quantum computing, circuit optimization, qubit, design automation.**

I. Introduction

Efficient quantum arithmetic circuits are fundamental to a wide range of quantum algorithms, from cryptographic applications to simulations in physics and chemistry, machine learning, and finance. For example, these circuits feature prominently in Shor's quantum factoring algorithm [1], whose implementation would render many cryptographic schemes, including RSA and ECC (elliptic curve cryptography), insecure. Despite the theoretical promise of speedups offered by quantum algorithms, practical implementations demand highly optimized arithmetic circuits to minimize qubit and gate resource overheads. At the core of quantum factoring algorithms there is a circuit for the modular exponentiation operation, which dominates the costs—including physical qubit count, gate count, and runtime—required for their execution. Hence, optimizing modular arithmetic subroutines within these algorithms is crucial for making quantum attacks on cryptography practically viable. In quantum computing, more significantly than in the classical case, circuit optimizations—such as reducing the number of gates or the space—can mean the difference between being able to run an algorithm and being unable to execute it at all. In this regard, significant progress has already been made in improving quantum arithmetic algorithms [2]–[6] and their applications, especially in cryptanalysis, e.g., with newer variants of factoring algorithms and more efficient modular arithmetic subroutines to reduce resource overheads [7]–[11]. One notable optimization is windowed quantum arithmetic, introduced by Gidney [12], which reduces the costs of modular multiplication by precomputing lookup tables and merging multiple operations into segments or "windows". This technique improves the scaling of modular exponentiation of n-bit numbers from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^3/\log^2 n)$, significantly reducing the overhead of modular exponentiation. The lookup–addition approach offloads some operations to a classical computer, thus optimizing quantum resource usage. Our work builds on these advancements by introducing four novel optimizations to windowed arithmetic, with a focus on cryptographically relevant instances.

Corresponding author: asireesh@ed.ac.uk

Through this work, we aim to emphasize the importance of enhancing existing subroutines in known quantum algorithms, as these optimizations significantly affect resource estimates for impactful applications.

Main results: In this paper, we study four improvements for windowed arithmetic circuits for modular multiplication and examine their impact on quantum algorithms for breaking RSA-2048. In Section III, we present four key algorithmic optimizations to the windowed arithmetic method, focusing on reducing the Toffoli cost and depth of memory lookups and uncomputation of lookups (unlookups), as well as minimizing the number of required lookups in the modular exponentiation algorithm used by Gidney–Ekerå (GE) [7]. First, we show how the cost of unlookups can be reduced by 50% asymptotically. We also illustrate how certain addresses can be bypassed, significantly reducing both circuit depth and the overall lookup cost. Furthermore, we demonstrate that by merging multiple lookup-addition operations into a single, larger lookup at the start of the modular exponentiation circuit, additional savings in both Toffoli gate count and circuit depth can be achieved.

In Section IV, to further showcase the impact of our optimizations, we study the improved physical qubit count, runtime, and error-correction overhead, providing updated resource estimates for attacking RSA-2048. Concretely, we test our improvements within the GE framework [7] and demonstrate a reduction in the computational volume for factoring RSA-2048 integers. Combined, these optimizations yield reductions in the Toffoli count for attacks against RSA by 1.5% to 3.4%, depending on the key size. These improvements help find parameters for the original GE algorithm that lead to a slight reduction in the overall computational volume for factoring RSA-2048. Specifically, we find that the time required decreases by about 16%, though with a 12% increase in the physical qubit count. The final two optimizations lead to a reduction in the number of *logical* qubits used in the windowing operation; however, we could not translate these reductions into a corresponding decrease in physical qubits or total runtime.

II. Preliminaries and Notation

We denote quantum registers as collections of qubits used for storing quantum information. For example, the state $|x\rangle_n$ refers to an n-qubit register in the computational basis with value $x \in \{0, 1, \ldots, 2^n - 1\}$. For windowed arithmetic, we divide a register $|x\rangle_n$ into a consecutive set of qubits or "windows" of size w qubits, with each window denoted as $x^{(i,w)}$ for $i = 0, 1, \ldots, \lceil n/w \rceil - 1$. Each $x^{(i,w)}$ is treated as an

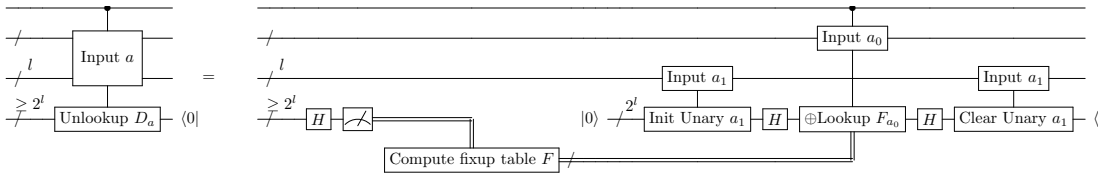


Fig. 1: Uncomputation of lookups i.e. unlookup circuit from [12] on a table of size $L = 2^l$, where l is the number of bits in the address register. Here the unlookup is performed on only half the address space, thus leading to a Tof gate count of $\mathcal{O}(\sqrt{L})$

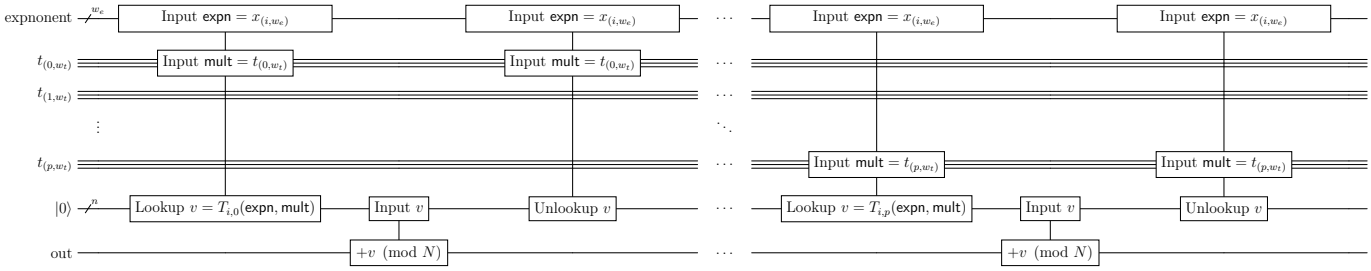


Fig. 2: Windowed modular exponentiation by [12]. Here, we are executing the modular multiplication of $b^{2^{iw_e} x^{(i,w_e)}}$ (the exponentiated value of the of the i^{th} window $x^{(i,w_e)}$ of the exponent), with the register $t = b^{x^{(0,iw_e)}}$ which is currently holding the exponentiation of the first $i-1$ windows of the exponent register.

independent w-bit value. For instance, the overall value of x can be expressed in terms of w-bit windows as follows:

$$x = \sum_{i=0}^{\lceil n/w \rceil - 1} x^{(i,w)} \cdot 2^{i \cdot w}.$$

A. Quantum table lookups

Efficient lookup operations form the basis of the the cost savings in windowed arithmetic. Quantum table lookups are a way to load classical or quantum data into memory [13]. For an address register $|a\rangle_l$, we have an associated lookup table of size $L = 2^l$. Let us assume that each memory element in the lookup table is of m bits, therefore, we need m qubits to store the value of the lookup. A lookup operation can be described as follows:

$$|a\rangle_l |y\rangle_m \xmapsto{\text{Lookup}} |a\rangle_l |y \oplus T_a\rangle_m$$

where T_a is the memory element in the lookup table for index a. A specific architecture for a quantum lookup table, termed a QROM lookup, was introduced in [13], requiring L Tof gates per lookup operation. The circuit for the QROM lookup can be seen in [12, Figure 2].

a) Uncomputation: motivation and improvements.: Uncomputation is often necessary in quantum algorithms to reset ancillary qubits and free up space for reuse. While Bennett's method [14] offers a general approach, it can double resource costs. Recent work [6], [12], [13], [15]–[17] has focused on reducing these overheads. For quantum table lookups, uncomputation (or unlookup) becomes critical to reset registers. Simplifications are possible, as shown by Gidney [12], leveraging a measurement-based uncomputation (MBU) technique. First, the lookup register is measured in the X basis, yielding a classical value s, which represents the measurement outcome. This operation can be represented as follows

$$\sum_a |a\rangle_\ell |T_a\rangle_m \xmapsto{\text{Measure}} \sum_a (-1)^{s \cdot T_a} |a\rangle_\ell |0\rangle_m ,$$

where s induces a phase based on T_a. Since the lookup table is known, the phase can be corrected using the following steps:

1. **Unary Conversion**: Split the address register a into a_{low} and a_{high} of u and $\ell - u$ qubits, respectively. Convert a_{low} into a unary basis:

$$\sum_a (-1)^{s \cdot T_a} |a_{\mathsf{high}}\rangle_{\ell-u} |a_{\mathsf{low}}\rangle_u |2^{a_{\mathsf{low}}}\rangle_{2^u} |0\rangle_{m-2^u} .$$

2. **Phase Correction Lookup**: Construct a simplified phase correction table $F_{a_{\mathsf{high}}}$ where $F(x) = 1$ for indices x satisfying $s \cdot T_{a_{\mathsf{high}} x} = 1$. Apply simultaneous phase corrections using the unary register as the lookup register.

3. **Clear Unary**: Perform an inverse unary operation to reset the unary register:

$$\sum_a |a\rangle_\ell |0\rangle_m .$$

The unlookup cost includes the unary conversion, phase correction lookup, and clearing the unary register. Using a temporary logical-AND, the total cost is $2^u + 2^{\ell-u}$ Tof gates. Optimal performance occurs when $u = \ell/2$, minimizing the cost to $2 \cdot 2^{\ell/2} = 2\sqrt{L}$, achieving a square-root speedup over the original lookup. This efficient unlookup operation is illustrated in fig. 1.

B. Windowed quantum arithmetic

The complexity of modular exponentiation can be reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^3/\log^2 n)$ by leveraging precomputed lookup tables. To prepare the state $\sum_x |x\rangle |b^x \mod N\rangle$, where x is a k-bit register, $b^x \mod N$ is expressed as:

$$b^x \mod N = \prod_{i=0}^{k-1} b^{2^i x_i} \mod N,$$

indicating that modular exponentiation consists of repeated controlled modular multiplications with controls on x_i. For each x_i, a controlled

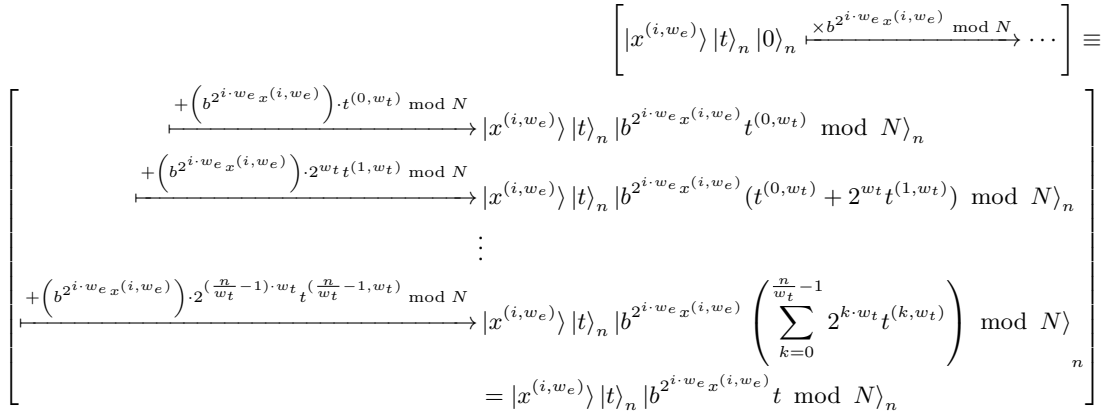


Fig. 3: An illustration of the windowing procedure by Gidney [12] for quantum modular exponentiation. $x^{(i,w_e)}$ represents the i-th window of size w_e in the exponent. $t^{(j,w_t)}$ represents the j-th window of size w_t in the multiplier t.

modular multiplication maps

$$|x_i\rangle_1 |t\rangle_n |0\rangle_n \xmapsto{\times b^{2^i x_i} \bmod N} \begin{cases} |x_i\rangle_1 |t\rangle_n |tb^{2^i} \mod N\rangle_n & \text{if } x_i = 1, \\ |x_i\rangle_1 |t\rangle_n |t\rangle_n & \text{if } x_i = 0. \end{cases}$$

This is further decomposed into modular additions for each bit t_j of t. Iteratively, we compute:

$$|x_i\rangle_1 |t\rangle_n |0\rangle_n \xmapsto{+\left(b^{2^i x_i}\right)\cdot 2^j t_j \bmod N} |x_i\rangle_1 |t\rangle_n |b^{2^i x_i} T \mod N\rangle_n,$$

where $T = \sum_{j=0}^{n} 2^j t_j$. The lookup table, indexed by (x_i, t_j), stores:

t_j	x_i	$(b^{2^i x_i}) \cdot 2^j t_j \mod N$
0	0	0
0	1	0
1	0	2^j
1	1	$2^j b^{2^i} \mod N$

TABLE I: Lookup table entries for a 2 bit lookup-addition based quantum modular multiplication.

This only exponentiates 1 bit of the exponent and has to be repeated k times for the k different bits of the exponent x. However, since all we are doing is performing table lookups followed by additions, we do not need to stop at just looking up 2 bits at a time. Let us assume we take a window w_e of bits for the exponent, and a window w_t of bits for our multiplication register t. This allows us to exponentiate w_e bits at a time as illustrated in figs. 2, 3.

The lookup table is now sized at $L = 2^{w_t + w_e}$ to store every possible combination of w_t and w_e-bit numbers, i.e. lookup value for every pair of t_j and x_i values, which is a product modulo N. Since each lookup and addition operation takes $\mathcal{O}(2^{w_e+w_t} + n)$ operations to retrieve, and we have to exponentiate every window of size w_e to the current window of size w_t, this leads to an asymptotic scaling of $\mathcal{O}(n^3/\log^2 n)$ if we take windows of size $w_e = w_t = \frac{1}{2}\log n$.

III. Improvements

In this section, we present several optimizations for windowed arithmetic circuits, particularly in scenarios involving multiple consecutive lookup operations. These optimizations focus on minimizing the number of Toffoli gates and circuit depth.

A. Deferred uncomputation for windowing circuits

While unlookups are much cheaper than their corresponding lookups, they can be made even cheaper in the multi-lookup setting (i.e. when more than one QROM lookup is involved) if part or all of the address register remains unchanged over many consecutive QROM lookups. This is the exact scenario in the case of the GE factoring algorithm, where for all the lookup-additions involved in multiplying $b^{x^{(i,w_e)}2^{iw_e}}$ (with index i, and window size w_e), the exponent window remains unchanged. These lookups have address $a = x^{(i,w_e)}||t^{(j,w_m)}$ with $j \in \left\{0, 1, \cdots, \frac{n}{w_m} - 1\right\}$, where $t^{(j,w_m)}$ is a window over the multiplication qubits. Each unlookup is of the form:

$$\begin{aligned} &\xmapsto{\mathsf{H,Measure}} (-1)^{S_j \cdot \ell_{i,j}} |x^{(i,w_e)}\rangle |t\rangle_n |S_j\rangle_n |r\rangle_n \\ &\xmapsto{\mathsf{Reset}\ S_j} (-1)^{S_j \cdot \ell_{i,j}} |x^{(i,w_e)}\rangle |t\rangle_n |0\rangle_n |r\rangle_n \\ &\xmapsto{\mathsf{Unary}} (-1)^{S_j \cdot \ell_{i,j}} |x^{(i,w_e)}\rangle |t\rangle_n \\ &\qquad |\mathsf{unary}(x^{(i,w_e)})\rangle_{2^{w_e}} |r\rangle_n |0\rangle_{n-2^{w_e}} \\ &\xmapsto{\mathsf{Lookup}\ F_{S_j}} |x^{(i,w_e)}\rangle |t\rangle_n \\ &\qquad |\mathsf{unary}(x^{(i,w_e)})\rangle_{2^{w_e}} |r\rangle_n |0\rangle_{n-2^{w_e}} \\ &\xmapsto{\mathsf{Clear\ Unary}} |t\rangle_n |0\rangle_n |r\rangle_n \end{aligned}$$

Each lookup is addressed by $a = \mathsf{exp}||\mathsf{mul}$ that combines an (outer loop) exponentiation window index exp and an (inner loop) addition window index mul. Gidney's [12] unary unlookup reduction is then applied to this construction.

We improve this further, exploiting the nested loop structure. Notice that the outer index exp stays fixed while the circuit iterates over the inner index mul. In our proposed optimization, we use mul as the address of the fixup table F_{S_j} and the unarized exp as the target. The binary-to-unary initialization of exp needs to be done only once before entering the inner loop, and the erasure of the unary register only once right at the end of the inner loop—whereas, in the unmodified version, there is a reset and initialization after every iteration within the inner loop. Thus, the cost of our circuit is dominated by the phase lookups (F_{S_j}) within each iteration, the one-time unary initialization and reset contributing negligibly in comparison. Overall, this leads to a halving of the Toffoli cost of unlookups in the GE factoring algorithm. After every lookup in the inner loop, let's instead perform

the operation (as shown in fig. 4)

$$\xmapsto{\text{H,Measure}} (-1)^{s_j \cdot \ell_{i,j}} |x^{(i,w_e)}\rangle |t\rangle_n |S_j\rangle_n |r\rangle_n$$
$$\xmapsto{\text{Reset } S_j} (-1)^{S_j \cdot \ell_{i,j}} |x^{(i,w_e)}\rangle |t\rangle_n |0\rangle_n |r\rangle_n$$

The value S_j is recorded classically, and a phase fixup table F_{S_j} is constructed (indexed/addressed by mul). At the end of the inner loop, we have $p = \left(\frac{n}{w_t} - 1\right)$ classical bit strings S_j, and p different phase fixup tables F_{S_j} ; $\forall j \in \{0, 1, \cdots, p-1\}$. The state at this stage is:

$$(-1)^{\sum_{k=0}^{p-1} S_k \cdot \ell_{i,j}} |x^{(i,w_e)}\rangle |t\rangle_n |0\rangle_n |r\rangle_n .$$

We can now construct a unary register with input $x^{(i,w_e)}$, and window over the register t to perform our phase corrections with F_{S_j}, i.e.

$$\xmapsto{\text{Lookup } F_{S_0}} (-1)^{\sum_{k=1}^{p-1} S_k \cdot \ell_{i,j}} |x^{(i,w_e)}\rangle |t\rangle_n |\mathsf{unary}(x^{(i,w_e)})\rangle_{2^{w_e}} |r\rangle_n |0\rangle_{n-2^{w_e}}$$

$$\xmapsto{\text{Lookup } F_{S_1}} (-1)^{\sum_{k=2}^{p-1} S_k \cdot \ell_{i,j}} |x^{(i,w_e)}\rangle |t\rangle_n |\mathsf{unary}(x^{(i,w_e)})\rangle_{2^{w_e}} |r\rangle_n |0\rangle_{n-2^{w_e}}$$

$$\vdots$$

$$\xmapsto{\text{Lookup } F_{S_{p-2}}} (-1)^{S_{p-1} \cdot \ell_{i,p-1}} |x^{(i,w_e)}\rangle |t\rangle_n |\mathsf{unary}(x^{(i,w_e)})\rangle_{2^{w_e}} |r\rangle_n |0\rangle_{n-2^{w_e}}$$

$$\xmapsto{\text{Lookup } F_{S_{p-1}}} |x^{(i,w_e)}\rangle |t\rangle_n |\mathsf{unary}(x^{(i,w_e)})\rangle_{2^{w_e}} |r\rangle_n |0\rangle_{n-2^{w_e}}$$

We now clear the unary register to retrieve our phase-free superposition

$$\xmapsto{\text{Clear Unary}} |x^{(i,w_e)}\rangle |t\rangle_n |0\rangle_n |r\rangle_n$$

B. Selective lookups

Although lookup tables differ for different moduli N and different exponent bases b, they exhibit similar structures for certain addresses. During a lookup addition of the i^{th} exponent window (of size w_e) and the j^{th} multiplication window (of size w_m), we construct a classical table $T_{i,j}$ corresponding to addresses stored in the register $m^{(j,w_m)}||x^{(i,w_e)}$. For a classical bit string address $a = \mathsf{mul}||\mathsf{exp}$, with $\mathsf{mul} \in \{0, 1, \cdots, 2^{w_m} - 1\}$ and $\mathsf{exp} \in \{0, 1, \cdots, 2^{w_e} - 1\}$, the lookup table $T_{i,j}$ in question holds the following values:

$$T_{i,j}(\mathsf{mul}, \mathsf{exp}) := (b^{\mathsf{exp}2^{iw_e}} 2^{jw_m} \mathsf{mul}) \mod N.$$

$T_{i,j}$ holds $2^{w_e+w_m}$ different values (for all combinations of length-w_e and -w_m bit strings). Inspecting the values associated with addresses where $\mathsf{mul} = 0$, we see that $T_{i,j}(\mathsf{mul}, *) = 0$ for all values of exp, regardless of the indices i and j. Therefore, we can start the lookups directly from $\mathsf{mul} = 1$ as we know that there is nothing to lookup for $\mathsf{mul} = 0$. This would save us 2^{w_m} lookups per query. On the other hand, for all addresses with $\mathsf{exp} = 0$, we get $T_{i,j}(\mathsf{mul}, \mathsf{exp}) = \mathsf{mul}2^{jw_m}$. Therefore, mul just gets copied directly to the target register. We propose to perform this copying at the beginning of every lookup. Note, the lookup table has to be altered now with lookup values

$$T'_{i,j}(\mathsf{mul}, \mathsf{exp}) = (b^{\mathsf{exp}2^{iw_e}} 2^{jw_m} \mathsf{mul} \mod N) \oplus 2^{jw_m} \mathsf{mul}.$$

We need to alter the table because the initial copying operation is an unconditional operation acting on all address inputs. We need to correct the lookup values where $\mathsf{exp} \neq 0$. The initial copying operation is analogous to the lookup $v = 2^{jw_m} \mathsf{mul}$. Using the corrected table, we find we now lookup $T'_{i,j}(\mathsf{mul}, \mathsf{exp})$, which alters the lookup register to

$$\begin{aligned} v &= 2^{jw_m} \mathsf{mul} \oplus T'_{i,j}(\mathsf{mul}, \mathsf{exp}) \\ &= 2^{jw_m} \mathsf{mul} \oplus (b^{\mathsf{exp}2^{iw_e}} 2^{jw_m} \mathsf{mul} \mod N) \oplus 2^{jw_m} \mathsf{mul} \\ &= (b^{\mathsf{exp}2^{iw_e}} 2^{jw_m} \mathsf{mul}) \mod N \\ &= T_{i,j}(\mathsf{mul}, \mathsf{exp}) \end{aligned}$$

This modification saves us $2^{w_m} - 1$ lookups (minus 1 because for the case $\mathsf{mul} = 0$, else we would be double counting the savings from the previous optimization). Refer to the circuit shown in fig. 5 for this copy-and-update-table lookup operation.

C. Larger initial lookup

While windowed modular exponentiation typically processes the exponent in small windows of size $\frac{1}{2} \log n$ to balance precomputation (i.e. size of lookups) and the number of iterations of the lookup-additions, we investigate an optimization for the initial phase of the algorithm. By combining multiple initial exponent windows into a single, larger lookup operation, we show below how to slightly reduce the number of modular multiplications and thus decrease the overall Toffoli gate count. Recall that with $x^{(i,w_e)}$ we represent the i-th window of size w_e in the exponent and with $t^{(k,w_t)}$ we represent the k-th window of size w_t in the multiplication register. We also recall from Section II-B how modular exponentiation is performed using windowing:

$$|x^{(0,w_e)}\rangle_{w_e} |1\rangle_n |0\rangle_{2n}$$
$$\xmapsto{\times b^{x^{(0,w_e)}} \bmod N} |x^{(0,w_e)}\rangle |b^{x^{(0,w_e)}}\rangle_n |0\rangle_{2n}$$
$$\xmapsto{\times b^{2^{w_e} x^{(1,w_e)}} \bmod N} |b^{x^{(0,2w_e)}}\rangle_n |0\rangle_{2n}$$
$$\vdots \text{ Iterate over } i \text{ exponent windows}$$
$$\xmapsto{\times b^{2^{i \cdot w_e} x^{(i,w_e)}} \bmod N} |b^{x^{(0,(i+1)w_e)}}\rangle_n |0\rangle_n$$

The first modular multiplication in the circuit can be reduced to a single lookup operation over the first $(i+1)w_e$ bits of the exponent. However, as i increases, this approach becomes more expensive than regular windowed modular multiplication, potentially negating the benefits of windowed arithmetic due to exponential growth in lookup table size. We explore the tradeoff between a direct lookup exponentiation for the first e bits of the exponent versus the cost of a lookup-addition based exponentiation.

D. Sliced windowing

During the windowing stage of the GE factoring algorithm, precomputed lookup table values are loaded into an n-qubit lookup register. We propose a method to reduce the number of logical qubits required for these lookups by effectively utilizing carry runways [18]. The GE algorithm employs carry runways to parallelize quantum addition. Assuming the carry runways divide the n-qubit quantum register into m pieces, each piece is of size $\frac{n}{m}$. Our approach involves loading only $\frac{n}{m}$ or a multiple of $\frac{n}{m}$ bits of the lookup table at a time. We then add this loaded register of size $\frac{n}{m}$ to the relevant quantum register, uncompute the values, and load the next $\frac{n}{m}$ bits. This strategy reduces the abstract qubit count of the GE factoring algorithm from $3n$ to $2n + \frac{n}{m}$. Although this increases the Toffoli count of the lookups by a factor of m, it can be managed by rearranging the order of the lookups and additions. We propose two variants of the sliced windowing scheme: Variant A, which reduces the logical qubit count

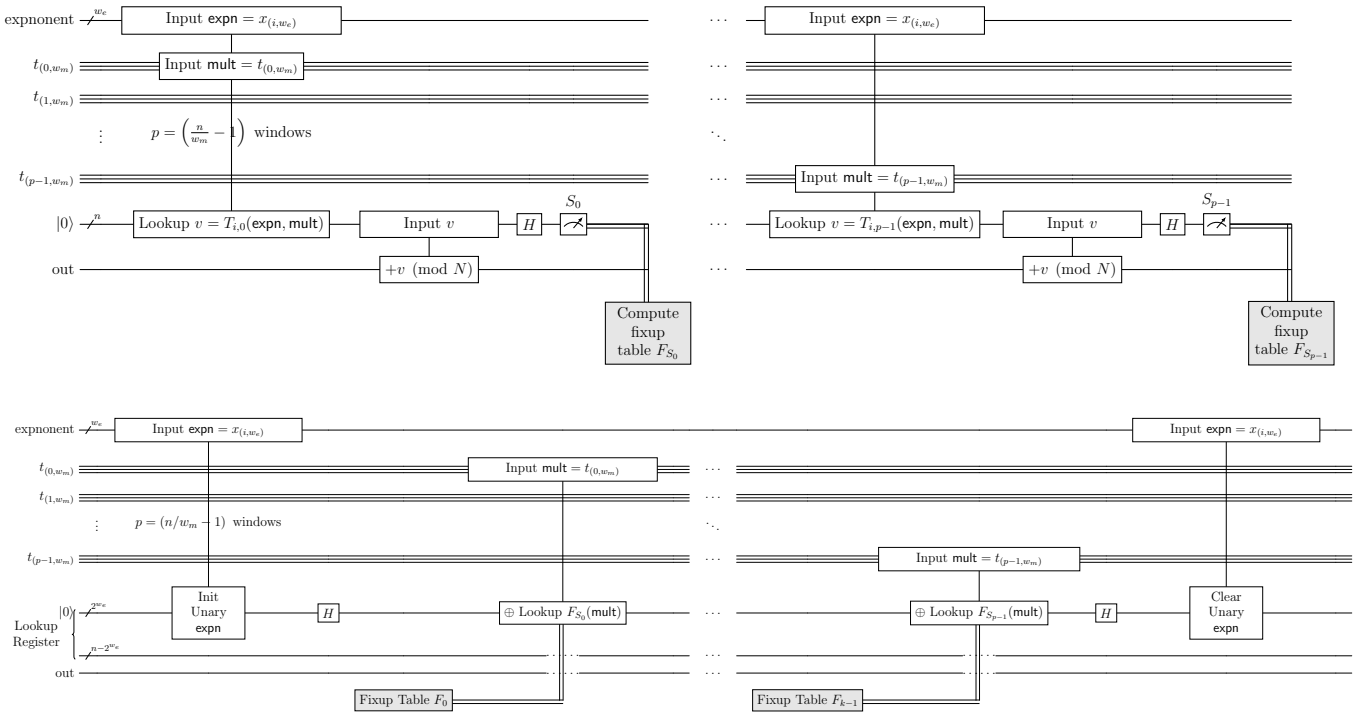


Fig. 4: Our proposed circuit for deferred uncomputation. The lookup additions have 2 stages. In stage one, the windowed multiplication with lookup qubit reset and no phase correction (top) is performed, followed stage 2 where the circuit for deferred phased correction (bottom) is executed. Notice that the unary conversion is performed only one, and on the exponent window as it is common for all lookups and can be reused. This is different from fig. 2 where the unlookups/phase corrections are performed after every single lookup addition.

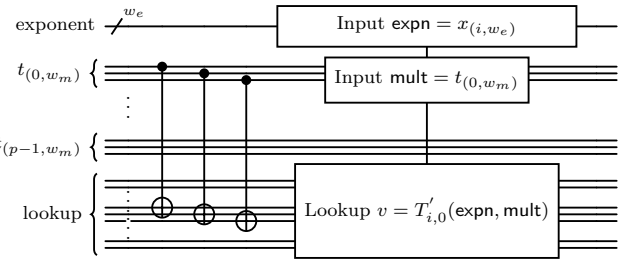


Fig. 5: Proposed optimization for pruned table. The CNOTs before the table lookup perform a copy operation, followed by an altered table lookup.

without impacting the Toffoli count, and Variant B, which reintroduces ancillas to lower the Toffoli count at the cost of increased circuit depth.

Variant A: Logical qubit reduction with sliced windowing: In this variant, lookup additions are performed on the first half of the target register $t[0 : n/2]$, while the second half $t[n/2 : n]$ serves as ancillas for Gidney's logical-AND adder. At each step, we only lookup the first $n/2$ bits into memory, compared to the full n bits in [12]. Each lookup costs $2^{w_e+w_m} = 2^{2\lfloor \frac{1}{2} \log n \rfloor}$ in Toffoli gates and depth. The addition of $n/4$-sized register pieces costs $n/2$ Toffoli gates, with depth n. In the second phase, another lookup-addition is performed with Cuccaro RCA addition (no ancillas), costing n Toffoli gates and depth n. Overall, this protocol does not reduce the Toffoli count but achieves a logical qubit reduction of $n/2$, with an increase in depth by n.

Variant B: Reintroducing temporarily protected ancillas for a lower Toffoli count: This variant reduces the Toffoli count by reusing ancillas that are idle during part of the computation. Instead of loading the entire lookup table, we process half at a time and repurpose ancillas saved in Variant A for addition into the second half of the target register. In Variant A, Gidney's adder [6] was used for the first half of the target register, with the second half acting as ancillas. For the second half, we switched to Cuccaro's ripple-carry adder due to the lack of ancillas. Here, we reintroduce the ancillas saved during the first stage, protecting them temporarily and "borrowing" them for Gidney's adder into the second half. This approach reduces the Toffoli count by $n/2$ while maintaining the same logical qubit count. However, it may increase circuit depth, as the ancillas must remain protected for longer, potentially offsetting the benefits in certain configurations.

IV. Application to breaking RSA-2048

In this section, we build upon the resource estimation framework introduced by Gidney and Ekerå [7] by incorporating our proposed enhancements to windowed arithmetic. Our analysis targets a superconducting qubit architecture, with computations encoded in rotated surface codes and logical operations performed using lattice surgery [19], [20]. We assume a physical error rate of 10^{-3} (and also test with a more optimistic 10^{-4} error rate), reflecting the fidelity of operations such as single- and two-qubit gates, state initialization, and measurements. Additionally, we consider a cycle time of 1μs, representing the duration required to measure all stabilizer of a surface code patch. These parameters are chosen to align with anticipated capabilities of future fault-tolerant quantum hardware and provide a consistent baseline for comparison. For a distance-d rotated surface code, we require $2(d+1)^2$ physical qubits to encode

n	n_e	**gate**$_{err}$	L_1	L_2	dev_{off}	g_{mul}	g_{exp}	g_{sep}	%	v.p.r	$\mathbb{E}$[vol]	Mqb	hrs	$\mathbb{E}$[hrs]	B Tofs
1024	1493	10^{-3}	15	27	5	5	5	1024	6%	0.507	0.539	9.624	1.264	1.344	0.407
2048	3029	10^{-3}	15	27	4	5	5	1024	31%	4.047	5.865	19.249	5.046	7.313	2.698
3072	4565	10^{-3}	17	29	6	4	5	1024	9%	18.328	20.141	37.897	11.607	12.755	9.885
4096	6101	10^{-3}	17	31	9	4	5	1024	5%	47.963	50.488	54.616	21.077	22.186	23.038
1024	1493	10^{-4}	7	13	4	5	5	512	5%	0.07	0.073	2.637	0.633	0.667	0.415
2048	3029	10^{-4}	7	13	4	5	5	512	21%	0.558	0.706	5.273	2.538	3.212	2.774
3072	4565	10^{-4}	9	15	5	5	5	768	2%	2.92	2.98	9.12	7.686	7.843	8.595
4096	6101	10^{-4}	9	15	5	4	5	512	3%	6.791	7.001	15.241	10.693	11.024	23.773
n	n_e	**gate**$_{err}$	L_1	L_2	dev_{off}	g_{mul}	g_{exp}	g_{sep}	%	v.p.r	$\mathbb{E}$[vol]	Mqb	hrs	$\mathbb{E}$[hrs]	B Tofs
1024	1493	10^{-3}	15	27	4	5	5	1024	6%	0.488	0.519	9.624	1.217	1.295	0.393
2048	3029	10^{-3}	17	27	6	5	5	1024	20%	4.419	5.524	21.616	4.906	6.133	2.656
3072	4565	10^{-3}	17	29	4	4	5	1024	9%	17.727	19.48	37.897	11.226	12.336	9.742
4096	6101	10^{-3}	17	31	8	4	5	1024	5%	46.424	48.867	54.616	20.4	21.474	22.8
1024	1493	10^{-4}	7	13	4	5	5	512	5%	0.067	0.071	2.637	0.612	0.644	0.402
2048	3029	10^{-4}	7	13	3	5	5	512	21%	0.541	0.684	5.273	2.461	3.115	2.72
3072	4565	10^{-4}	9	15	5	5	5	768	2%	2.851	2.909	9.12	7.503	7.656	8.486
4096	6101	10^{-4}	9	15	5	4	5	512	3%	6.588	6.792	15.241	10.374	10.695	23.559

TABLE II: Comparison of resource estimates for the GE factoring algorithm without improvements (top) and with our proposed optimizations (bottom). The columns are: the number of bits of the number to factor (n), the bits in the exponentiation register (n_e), the gate error rate ($gate_{err}$, e.g., 10^{-3}), the distances for the first and second layers of error correction (L_1 and L_2), the deviation due to padding error in coset representation (dev_{off}), the window size for the multiplication register in windowed arithmetic (g_{mul}), the window size for the exponentiation register in windowed arithmetic (g_{exp}), the separation in oblivious ripple carry adders (g_{sep}), the retry risk (probability of error during computation, as a percentage), the computation volume for a single run expressed in megaqubit-days (v.p.r.), the expected computation volume accounting for retry risk ($\mathbb{E}$[volume]), the number of logical qubits in millions (megaqubits, Mqb), the runtime for a single run in hours (Runtime), the expected runtime accounting for errors ($\mathbb{E}$[hours]), and the total number of Toffoli gates in billions (B Tofs)

a single logical qubit of information. A code distance of $d = 27$ is proposed as sufficient for factoring RSA-2048 on large-scale quantum hardware, balancing the trade-off between logical error rates and resource requirements. Our goal is to minimize the *skewed volume*, a key metric that balances the trade-offs between qubit resources and runtime, defined as:

$$\text{skewed_volume} = (\text{Mqb})^{1.2} \times \mathbb{E}[\text{hrs}]$$

where Mqb is the total number of physical qubits in megaqubits (millions of qubits), and $\mathbb{E}$[hrs] is the expected runtime in hours, accounting for the likelihood of retries due to logical errors, with the exponent of 1.2 reflecting a preference for reducing qubit (Mqb) over runtime ($\mathbb{E}$[hrs]), and can be varied based on the specific use cases.

The improvements described in Sections III-A, III-B, and III-C focus on reducing the Toffoli count and computational volume for modular arithmetic operations. When combined, these optimizations yield reductions in the Toffoli count and depth for cryptographically relevant attacks on RSA factoring by 1.5% to 3.4%, as summarized in Table II. More specifically, we also see nearly a 16% reduction in the expected runtime of the algorithm for factoring RSA-2048 bit integers at the cost of a 12% increase in the physical qubit count (owing to the reduced retry risk of the algorithm). The sliced windowing technique, as described in Section III-D, offers a trade-off at a logical level. While it reduces the number of logical qubits required, it comes at the expense of an increased circuit depth. This added depth necessitates higher-distance QEC codes to ensure fault tolerance, which can lead to a net increase in the overall resource cost, and hence is not included in our comparison. Whether sliced windowing can be used to reduce the number of physical qubits for a particular kind of hardware architecture and an error correction code is a non-trivial question, which is left for future research.

The techniques presented in this work, while demonstrated through their application to improve the efficiency of factoring algorithms, have broader utility. They can be applied to simplify and optimize algorithms that rely on (windowed) arithmetic, thereby improving both performance and scalability across a wider range of computations.

ACKNOWLEDGEMENTS

This work started when AS was working at Inveriant Pte. Ltd. AS is supported by Innovate UK under grant *10004359*. This work is supported by the National Research Foundation, Singapore, and A*STAR under its Centre for Quantum Technologies Funding Initiative (S24Q2d0009). We also acknowledge funding from the Quantum Engineering Programme (QEP 2.0) under grants *NRF2021-QEP2-02-P05* and *NRF2021-QEP2-02-P01*. We thank Anupam Chattopadhyay for useful discussions on quantum arithmetic. We thank Filippo Miatto, Ilan Tzitrin, and Rafael Alexander for useful discussions on resource estimations on photonic architectures, and Miklos Santha useful discussions on space-time trade-offs in arithmetic.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[2] T. G. Draper, "Addition on a quantum computer," *arXiv preprint quant-ph/0008033*, 2000.

[3] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," *arXiv preprint quant-ph/0410184*, 2004.

[4] G. D. Kahanamoku-Meyer and N. Y. Yao, "Fast quantum integer multiplication with zero ancillas," *arXiv preprint arXiv:2403.18006*, 2024.

[5] D. Litinski, "Quantum schoolbook multiplication with fewer toffoli gates," *arXiv preprint arXiv:2410.00899*, 2024.

[6] C. Gidney, "Halving the cost of quantum addition," *Quantum*, vol. 2, p. 74, 2018.

[7] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.

[8] D. Litinski, "How to compute a 256-bit elliptic curve private key with only 50 million toffoli gates," *arXiv preprint arXiv:2306.08585*, 2023.

[9] É. Gouzien, D. Ruiz, F.-M. Le Régent, J. Guillaud, and N. Sangouard, "Performance analysis of a repetition cat code architecture: Computing 256-bit elliptic curve logarithm in 9 hours with 126 133 cat qubits," *Physical Review Letters*, vol. 131, no. 4, p. 040602, 2023.

[10] C. Chevignard, P.-A. Fouque, and A. Schrottenloher, "Reducing the number of qubits in quantum factoring," *Cryptology ePrint Archive*, 2024.
[11] S. Wang, X. Li, W. J. B. Lee, S. Deb, E. Lim, and A. Chattopadhyay, "A comprehensive study of quantum arithmetic circuits," *arXiv preprint arXiv:2406.03867*, 2024.
[12] C. Gidney, "Windowed quantum arithmetic," *arXiv preprint arXiv:1905.07682*, 2019.
[13] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven, "Encoding electronic spectra in quantum circuits with linear t complexity," *Physical Review X*, vol. 8, no. 4, p. 041015, 2018.
[14] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of R&D*, vol. 17, no. 6, pp. 525–532, 1973.
[15] C. Jones, "Low-overhead constructions for the fault-tolerant toffoli gate," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 87, no. 2, p. 022328, 2013.
[16] N. Kornerup, J. Sadun, and D. Soloveichik, "Tight bounds on the spooky pebble game: Recycling qubits with measurements," *arXiv preprint arXiv:2110.08973*, 2021.
[17] A. Luongo, A. M. Miti, V. Narasimhachar, and A. Sireesh, "Measurement-based uncomputation of quantum circuits for modular arithmetic," *arXiv preprint arXiv:2407.20167*, 2024.
[18] C. Gidney, "Approximate encoded permutations and piecewise quantum adders," *arXiv preprint arXiv:1905.08488*, 2019.
[19] D. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, vol. 14, no. 12, p. 123011, 2012.
[20] A. G. Fowler and C. Gidney, "Low overhead quantum computation using lattice surgery," *arXiv preprint arXiv:1808.06709*, 2018.

VISTA: Optimizing GPU Scheduling through Versatile Locality-Aware Data Sharing

Hajar Falahati*, Negin Mahani*, Adrian Cristal, and Osman Unsal
Barcelona Supercomputing Center (BSC), Barcelona, Spain
{hajar.falahati, neginsadat.nematollahizadeh, adrian.cristal, osman.unsal}@bs.es

***Abstract*—Graphics Processing Units (GPUs) play a pivotal role in high-performance computing by utilizing the massive parallelism of concurrent thread execution to enhance processing efficiency, known as thread-level parallelism (TLP). However, effectively managing the significant memory access generated by a large number of concurrent threads presents a significant challenge, affecting both performance and energy consumption. However, previous GPU scheduling methods often overlook the significant potential of data sharing among non-adjacent warps or Cooperative Thread Arrays (CTAs), thereby limiting their effectiveness in reducing unnecessary memory accesses through supporting data sharing. Our observation underscores the untapped potential of non-adjacent data sharing, motivating our work to optimize schedules more effectively. In this paper, we present VISTA, a smart locality-aware GPU scheduler that dynamically identifies data locality patterns at both the CTA and warp levels, leveraging this runtime information to make intelligent scheduling decisions and improving memory efficiency and overall performance. VISTA significantly reduces unnecessary memory accesses. Simulation results show a 48.1% improvement in performance and a 51.8% reduction in energy consumption for memory-intensive GPGPU applications, compared to the baseline, with negligible hardware overhead.**

***Index Terms*—GPGPU, Memory Access, Energy Efficiency, Performance, Data Sharing, Scheduling.**

I. Introduction

The rising demand for applications with intensive processing requirements has led to a surge in the utilization of Graphics Processing Units (GPUs). GPUs offer remarkable throughput by leveraging high Thread Level Parallelism (TLP), but deploying general-purpose applications on these processors presents significant memory challenges [1], [2].

While increasing TLP aims to hide latency, it also intensifies simultaneous memory access, leading to cache contention and bandwidth saturation. A significant portion of performance degradation comes from L1D cache misses, which lead to accesses to the L2 cache and DRAM, exacerbating power consumption and latency due to excessive memory hierarchy and interconnection network accesses [3]–[7].

Previous work to tackle memory access challenges in GPUs mainly focuses on refining scheduling mechanisms. Various scheduling policies have been proposed to fine-tune memory access patterns and enhance overall system performance [8]–[23]. These policies typically aim to exploit data locality within the GPU's memory hierarchy, striving to minimize unnecessary memory accesses and cache misses. However, existing scheduling methods fail to effectively capture and utilize data sharing among threads. For example, conventional schedulers typically employ round-robin or throttling-based strategies. Meanwhile, contemporary warp or Cooperative Thread Arrays (CTA) scheduling techniques [15], prioritize preserving locality in L1D cache. They achieve this by scheduling neighboring warps or CTAs on the same Streaming Multiprocessors (SM) and granting them execution priority. However, this approach may not fully use the opportunity for data sharing among non-adjacent CTAs or warps (i.e., CTAs or warps with IDs differing by more than 1).

Through our analysis of data-sharing patterns in GPUs, we make three key observations that underscore the potential for improving memory access efficiency. First, memory-intensive GPGPU applications exhibit significant data sharing, 80.4% on average within a single SM (Intra-SM) and 68.2% between different SMs (Inter-SM). The substantial Inter-SM data sharing points to inefficiencies in current CTA scheduling policies, which fail to fully exploit existing data sharing. Second, we find that 95% to 78% of shared data occurs as the distance between SMs (i.e., the difference between two SM IDs) increases from 0 to 40, indicating substantial data sharing between non-adjacent CTAs. Third, we observe that non-adjacent warps account for around 80% of total data sharing within an SM, underscoring the opportunity to optimize memory access by capturing this sharing. However, existing sharing-based schedulers overlook the benefits of non-adjacent data sharing. This highlights an urgent need to explore and exploit data sharing among non-adjacent CTAs and warps to mitigate memory access challenges in GPUs.

We propose VISTA, a smart locality-aware GPU scheduler that dynamically identifies data locality patterns at both the CTA and warp levels using a lightweight learning-based model to decrease unnecessary memory accesses while preserving L1D cache locality.

First, VISTA proactively identifies data locality among CTAs, classifies CTAs based on locality, and assigns them to the SM with the most similar memory behavior. To achieve this, VISTA features a CTA locality tracker that utilizes a lightweight clustering approach to group CTAs based on their memory access patterns. It monitors each CTA's memory behavior by running sample warps on idle subcores and assigns the CTA to the SM with the most similar memory access patterns (i.e., those that access the same addresses in the same order).

*These authors contributed equally to this work.

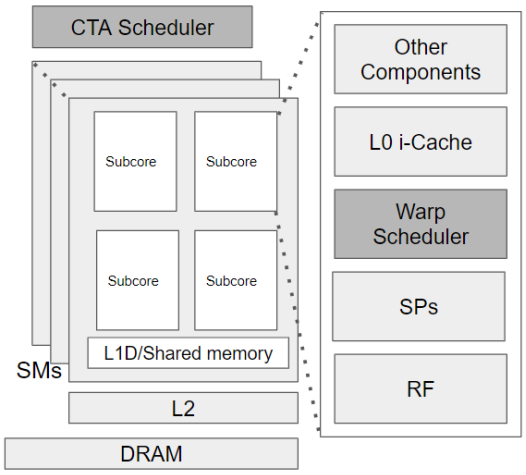


Fig. 1. Baseline GPU architecture.

Second, once CTAs are assigned to the SMs with the most similar memory access patterns, VISTA further detects data locality among warps and prioritizes those more likely to reuse data for execution. This is achieved by leveraging both the history of application behavior and the runtime memory access patterns of the warps. To achieve this, incorporates a warp locality tracker built on a lightweight learning-based model, the Integer Support Vector Machine (ISVM). The warp locality tracker leverages the ISVM model to analyze the history of program counters from memory instructions and tracks the runtime behavior of warp accesses. This unit also generates a hashed vector that represents the memory pattern of each SM.

In this paper, we make the following contributions:

- We observe that previous scheduling methods fall short as they fail to leverage data sharing among non-adjacent CTAs or warps.
- We introduce VISTA, a novel approach that harnesses data locality among adjacent and non-adjacent CTAs and warps, incorporating dynamic locality-aware CTA and warp schedulers.
- Results show that VISTA improves performance by 48.1% and reduces energy consumption by 51.8% for memory-intensive GPGPU applications, compared to the baseline, with negligible hardware overhead.

II. Background and Motivation

Figure 1 provides an overview of the GPU architecture. GPUs consist of multiple SMs, with a global Giga Thread scheduler that distributes CTAs to SMs using a round-robin policy. Within each SM, threads execute in the form of warps (a group of threads executing in lock-step), which are scheduled by warp schedulers. Each SM comprises various key units, such as Streaming Processors (SP) grouped in four subcores, Load Store Units (LD/ST), warp schedulers, and on-chip storages (e.g., register files and L1D cache). SMs are connected to the memory hierarchy and off-chip DRAM through a network. The memory hierarchy includes L2 cache banks, memory controllers, and DRAM channels.

TABLE I
Percentage of data sharing in GPGPU benchmarks.

Benchmark suite	Intra-SM Data sharing%	Inter-SM Data sharing%
Ispass-2009 [29]	81%	68%
Rodinia [30]	65%	53%
LonestarGPU [31]	85%	72%
PolyBench [32]	89%	78%
Parboil [33]	63%	52%
Pannotia [34]	84%	71%
GraphBench [35]	87%	70%
SDK [36]	78%	69%
DeepBench [37]	92%	81%

The memory hierarchy, combined with high TLP, is designed to meet the extensive memory demands of memory-intensive GPGPU applications.

A major bottleneck in memory-intensive GPGPU applications is the large number of memory accesses generated by thousands of threads, leading to frequent cache contention and evictions. An effective approach to mitigate this issue is to preserve data locality by retaining shared data for reuse across threads, which reduces the frequency of memory requests [24]. Thus, closely related CTAs are assigned to the same SM, significantly improving memory system performance.

We analyze the extent of data sharing across various levels and investigate sharing patterns in memory-intensive GPGPU applications. For more details about our methodology, refer to §IV. To illustrate the prevalence of such accesses in memory-intensive GPGPU applications, we measure the fraction of memory accesses shared within an SM and across SMs in GPGPU applications. Table I summarizes these results. We define a memory access as shared whenever a thread accesses an address previously accessed by another thread. Our observations show that all applications exhibit significant inter-SM and intra-SM data sharing, with 68.2% and 80.4% of memory accesses shared, respectively.

To analyze data-sharing patterns, we quantify the shared data among threads based on SM distance, as shown in Figure 2. Our results show that 95% to 78% of the shared data occurs as the SM distance increases from 0 to 40, highlighting significant data sharing between non-adjacent CTAs. We further observe that around 80% of the total data sharing within an SM occurs between non-adjacent warps.

While previous studies have confirmed notable data sharing in GPUs [1], [3], [25]–[28], they have mainly focused on adjacent CTAs or warps, or on sharing across SMs. These approaches fail to fully exploit the potential of dynamically scheduling both warps and CTAs. In contrast, our findings emphasize the importance of data sharing among non-adjacent CTAs and warps, which can be achieved by proactively scheduling these high-locality CTAs and warps closer together.

To summarize, the increased frequency of memory requests leads to higher delays and power consumption within the GPU memory subsystem. Even in high TLP scenarios, where numerous parallel threads generate thousands of memory accesses, significant contention occurs which further exacerbates delays, especially due to queuing delays in memory resources [25], [27]. In this work, we build upon the key observation that non-adjacent CTAs and warps share a sub-

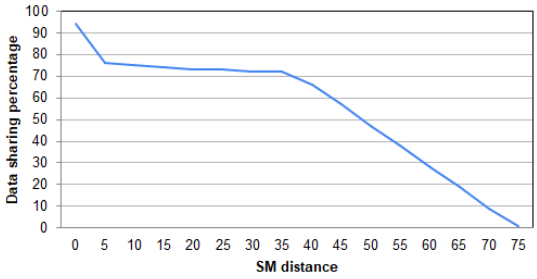


Fig. 2. Data-sharing pattern with respect to SM distance.

stantial amount of data. Existing scheduling mechanisms fail to simultaneously leverage data locality within SMs and data sharing across CTAs, missing opportunities to fully optimize data sharing in GPUs. Our goal is to propose a proactive scheduling mechanism for GPUs that addresses these gaps by 1) detecting and utilizing data sharing among non-adjacent warps within each SM while preserving cache locality, and 2) discovering and leveraging data sharing among non-adjacent CTAs. We believe that integrating these two features in scheduling improves overall performance and reduces energy consumption in GPUs.

III. VISTA

We propose VISTA, a smart locality-aware scheduler for GPUs, designed to dynamically detect and exploit fine-grained data locality patterns at both CTA and warp levels through intelligent scheduling decisions. By doing so, VISTA optimizes memory management, reduces memory accesses, and significantly enhances performance and energy efficiency.

First, VISTA dynamically identifies data locality among CTAs by running two warps of unscheduled CTAs on idle SMs. By executing the first few memory instructions (typically 4) of two representative warps, VISTA tracks the memory behavior of their associated CTAs. It then classifies CTAs based on locality and assigns them to the SM with the most similar memory access pattern. Once CTAs are grouped and assigned to the SMs with the most similar memory access patterns, VISTA further enhances memory usage by dynamically detecting data locality among warps using a lightweight learning-based model. The warp scheduler is modified to prioritize warps that are more likely to reuse data.

VISTA consists of two key units: the warp locality tracker and the CTA locality tracker. First, we describe the warp locality tracker and the process of extracting identifiers for memory patterns of SMs (§III-A). Second, we describe how the CTA locality tracker (§III-B) classifies incoming CTAs based on the similarity of their memory access pattern to previously observed patterns in each SM.

A. Warp Locality Tracker

Each SM contains a warp locality tracker that utilizes a lightweight learning-based model, an Integer Support Vector Machine (ISVM) [38], to predict the memory behavior of the assigned warps. By analyzing the memory pattern of warps assigned to the SM, the warp locality tracker (1) identifies which warps have a higher likelihood of maintaining data locality within the SM and gives these warps higher execution priority, and (2) creates a hashed vector representing the SM's memory pattern (Pattern SM), used to detect incoming CTAs with similar memory access patterns.

The warp locality tracker is composed of two main components: a prediction unit, which forecasts memory behavior and data locality, and a priority unit, which assigns execution priority to warps based on these predictions.

1) Prediction Unit: Figure 3 and Figure 4 depict the prediction unit structure and its flowchart, respectively. To predict the memory behavior among warps and CTAs, we (1) utilize the ISVM model that considers the history of program counters from memory instructions (PC_{mem}), and (2) consider the runtime behavior through tracking the history of warps accesses. Consistent with prior research [38], [39], we found it unnecessary to consider a full instruction history. Consequently, our designed ISVM model tracks the 16 recent warp accesses working on selected features extracted from recent PC_{mem} and data sharing percentage history. We train the ISVM model by profiling data (logged PC_{mem} and data sharing percentage) from a diverse set of applications, focusing on 20% of the representative GPGPU kernels.

Warp locality prediction. Warp locality prediction consists of two key units: the PC history and the sharing history units. The PC history unit includes a hashed PC table, a weight table, and a hashing mechanism (F_{hash} in Figure 3). The hashed PC table stores 16 hashed PCs, each pointing to an entry in the sharing history unit. The hashing mechanism creates a 4-bit vector by XOR-ing 16-bit chunks of the 64-bit PC_{mem}. The weight table holds trained weights for each hashed PC pattern, which quantifies the contribution of each PC to the warp locality score. The sharing history unit monitors which warp accesses each hashed memory address. It is organized as a 16-entry table, with each entry representing a hashed memory address. Each entry contains 64 elements, one for each ready warp, and tracks up to 16 memory accesses per warp. To calculate the sharing score for each warp, the total number of memory accesses issued by that warp is divided by the total number of accesses, producing a score between zero and one ($F_{sharing_score}$ in Figure 3).

When a new PC_{mem} is encountered, it is hashed and used to index the hashed PC table. If the pattern is new, a corresponding entry is allocated in the sharing history unit. For previously seen patterns, the sharing history unit is updated. The weights and sharing scores (S_i in Figure 3) for each hashed PC pattern are multiplied to generate the overall warp locality score (FS_i).

SM memory pattern. The pattern detector unit selects the four hashed memory PC with the highest sharing amount. The sharing amount is determined by counting how many accesses are issued by different warps. To represent the overall memory behavior of each SM (SM memory pattern), the pattern detector generates a compact 16-bit signature vector consisting of four 4-bit hashed memory PCs, indicating which hashed memory addresses are most frequently accessed.

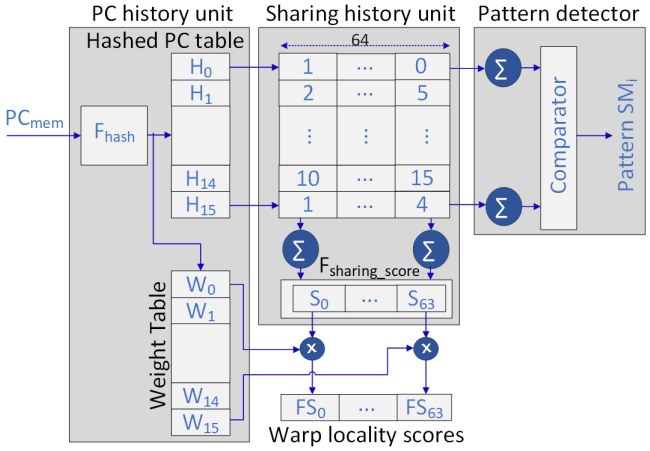


Fig. 3. Structure of prediction unit in warp locality tracker.

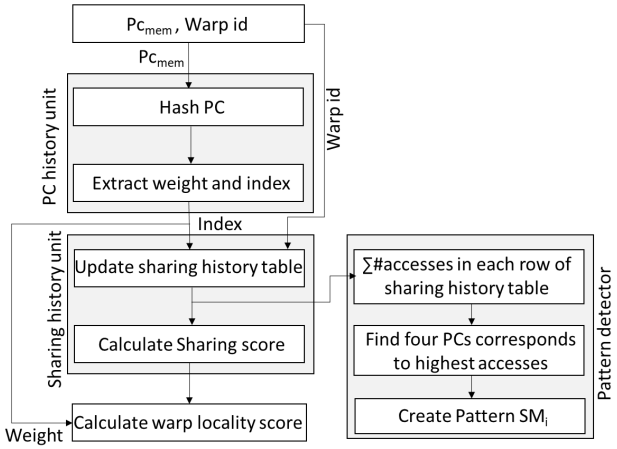


Fig. 4. Flowchart of prediction unit in warp locality tracker.

2) Priority Unit: Based on the detected memory access patterns, the priority unit selects the warps with the highest potential for maintaining data locality, as shown in Figure 5. It does this by evaluating eligible warps as execution candidates and sorting them according to their warp locality scores (FS$_i$ in Figure 3). Warps with the highest locality scores are given priority for execution. In cases where multiple warps have the same score, the priority unit breaks the tie by selecting the warp with the lowest ID.

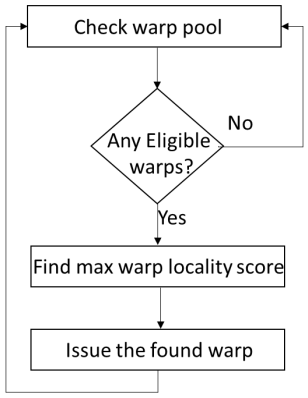


Fig. 5. Priority flowchart.

B. CTA Locality Tracker

We enhance the GPU's CTA scheduler with a CTA locality tracker, which aims to assign each incoming CTA to the SM that exhibits the most similar memory access pattern. The CTA locality tracker employs Locality-Sensitive Hashing (LSH) for similarity detection. LSH is an efficient online clustering technique known for its low overhead and high clustering accuracy. Figure 6 shows the LSH-based similarity checker. It extracts the memory access pattern of a CTA, compares it with the memory patterns of SMs, and selects the SM with the closest matching memory pattern for the CTA assignment.

CTA access pattern. To capture the memory access pattern of a CTA, the CTA locality tracker executes its first warp and one randomly selected warp until four memory accesses are issued in these two warps. Prior research suggests that observing four memory accesses provides a good approximation of the overall memory behavior of a warp [39]. These warps are executed on idle subcores, as there are typically idle subcores in GPUs [40].

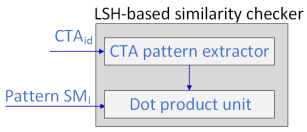


Fig. 6. CTA locality tracker structure.

Using LSH, the accessed memory addresses are projected onto a lower-dimensional binary space via multiple hash functions, forming a 16-bit binary locality signature, similar to the way ISVM handles memory addresses in the warp locality tracker (F$_{hash}$ in Figure 3). Each warp's memory accesses are hashed into this 16-bit vector, and by XOR-ing the vectors from the two executed warps, the CTA locality tracker derives a 16-bit vector that represents the CTA's access pattern.

Assigning CTA. Figure 7 shows how VISTA assigns CTAs to the best match SM. Once the CTA access pattern is extracted, the CTA locality tracker compares it with the memory patterns of all SMs to find the best match. The similarity between the CTA's memory pattern and each SM's pattern is measured by calculating the distance between their respective signature vectors. The CTA is then assigned to the SM with the smallest distance. In cases where multiple SMs exhibit similar distances, the tracker resolves this by selecting the SM with the lowest load. To calculate the similarity score, the CTA locality tracker performs a dot product between the CTA memory pattern and the memory patterns of the SMs. The SM with the highest similarity score is chosen as the destination for the CTA.

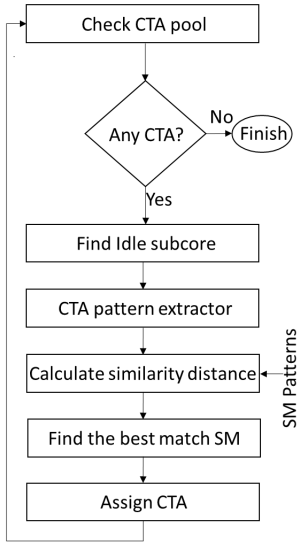


Fig. 7. CTA locality tracker flowchart.

IV. Methodology

Simulation. We evaluate VISTA using Accel-Sim V1.0 [41] and Accel-Wattch [42] to model an NVIDIA Volta QV100 GPU [43]. Table II shows simulation parameters. We modify the CTA scheduler and warp scheduler to provide the architectural supports in our proposal (§III). The reported results fully incorporate the overheads introduced by VISTA.

Workloads. We evaluate our proposal under a wide range of standard GPGPU applications from Rodinia [30], LonestarGPU [31], Ispass2009 [29], Pannotia [34], and Graph-Bench [35] benchmark suites (Table III). We selected representative benchmarks from various domains, 40 applications for training, and 20 distinct applications for testing, achieving an average data locality prediction accuracy of 85% on the training set and 80% on the test set. We further validate the robustness of the model by evaluating it on memory-intensive workloads excluded from the training set.

TABLE II
SIMULATION CONFIGURATION.

Parameter	Value
Number of SM	80
Core clock	1530 MHz
Scheduler	Greedy-then-Oldest
Number of scheduler per SM	4
Number of threads per SM	2048
Register file size per SM	65536
Unified cache	adaptive, 128KB, 256-way, 128B-line, 4-bank, 28-cycle
L1 instruction cache	128KB, 16-way, 128B-line
Constant cache	64KB, 8-way, 64B-line
L2 cache	96 KB/subpartition, 24-way, 128B-line, 32-set, 160-cycle

TABLE III
BASELINE BENCHMARK SUITE.

Application	Abbreviation
Breadth-first search	BFS [35]
B+tree search	BTR [30]
Hotspot	HSP [30]
Needleman-wunsch	NW [30]
LU decomposition	LUD [30]
Speckle Reducing Anisotropic Diffusion	SRAD [30]
Ray tracing	RAY [29]
StoreGPU	STO [29]
Components Labeling	CCL [31]
Particle filter	PF [30]
Page rank	pr [34]

Comparison points and metrics. We compare VISTA with the following systems: (1) Baseline: an NVIDIA Volta QV100 [43], configured as in Table II); (2) CTA-VISTA: a variant of VISTA without the warp locality tracker unit; (3) Warp-VISTA: a version of VISTA without the CTA locality tracker unit; (4) CCDS [3]: a state-of-the-art data-sharing-aware SM coordination system; and (5) CCWS [15]: a locality-aware warp scheduler. We report the instruction per cycle (IPC) and the energy consumption as the comparison metrics.

V. EVALUATION

Performance and energy analysis. Figure 8 presents the performance of all comparison points, normalized to the Baseline. We draw two key observations. First, all evaluated designs (VISTA, CTA-VISTA, Warp-VISTA, CCDS [3], and CCWS [15]) outperform the Baseline for memory-intensive GPGPU applications, delivering average IPC gains of 48.1%, 33.8%, 24.1%, 21.2%, and 12.4%, respectively. Second, applications with high memory intensity and non-adjacent data sharing patterns, such as PF [30], benefit significantly more from VISTA compared to the other designs.

Figure 9 reports the energy consumption normalized to Baseline. We observe that VISTA, CTA-VISTA, Warp-VISTA, CCDS [3], and CCWS [15] reduce the total energy consumption of memory-intensive GPGPU applications by an average of 51.8%, 39.7%, 30.1%, 22.8%, and 7.5%, respectively.

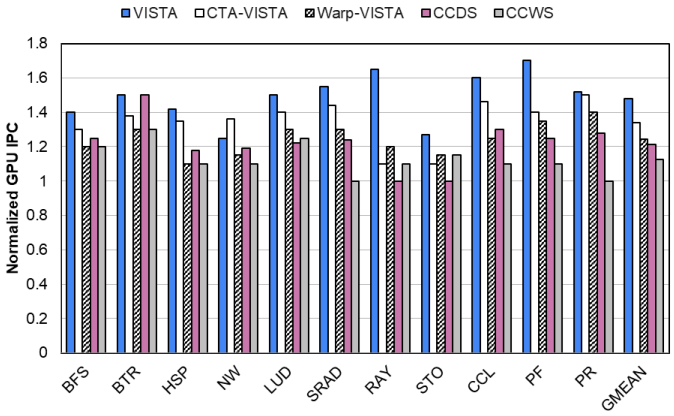


Fig. 8. Total GPU IPC of VISTA, CTA-VISTA, Warp-VISTA, CCDS, and CCWS normalized to Baseline.

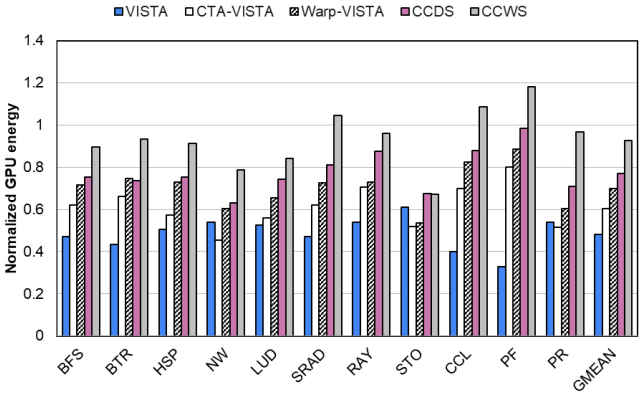


Fig. 9. Total GPU energy consumption of VISTA, CTA-VISTA, Warp-VISTA, CCDS, and CCWS normalized to Baseline.

Source of benefits. To better understand the advantages of our proposed approach, we analyze how VISTA impacts the L1D/L2 cache hit rate. Figure 10 reports the L1D/L2 hit rate improvements of VISTA normalized to Baseline. We observe that VISTA improves the L1D and L2 hit rates by 53.9% and 48.7%, respectively, reduces memory stalls by 50.4%, and decreases execution stalls by 46.3%, compared to Baseline for memory-intensive GPGPU applications. These results demonstrate that VISTA effectively improves data locality and execution efficiency, thereby reducing both memory and scheduling stalls.

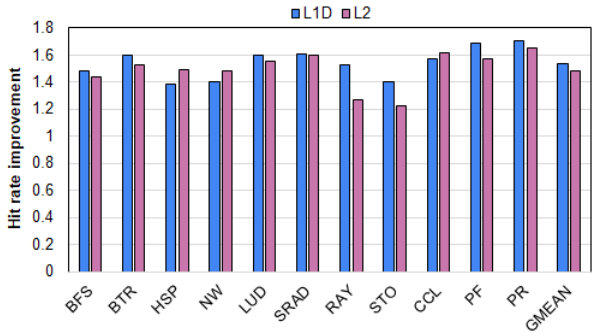


Fig. 10. L1D/L2 Hit rate improvement of VISTAnormalized to Baseline.

Hardware Overhead. VISTA introduces two key components: a warp locality tracker for each SM and a CTA locality tracker for the entire GPU. These components are evaluated for area and power overheads using a hybrid modeling approach that combines detailed architectural simulations with analytical modeling for the unchanged portions of the GPU. The warp locality tracker consists of four main units. (1) Hashed PC table and sharing history table consist of 16 and 64 $\times$ 16 entries (4 bits per entry), requiring 8B and 128B of storage, respectively. (2) Weight table, F_sharing-score, and warp locality score with 16, 64, and 64 entries, respectively, where each element is 8 bits; The total storage for these components is 16 B, 64 B, and 64 B, respectively. (3) The logic unit includes a hash function and a total of 2,112 4-bit adders, comprising 64 $\times$ 16 adders for the hashed PC and another 64 $\times$ 16 for sharing history operations, and 64 additional 8-bit adders for score computation. (4) The CTA locality tracker includes 16 multipliers and 15 adders, working on 16-bit operands.

Area. The storage components for the warp and CTA locality trackers are modeled using CACTI V7.0 [44] for 22nm technology and scaled to a 12nm process node, consistent with the fabrication technology of the NVIDIA Volta V100 GPU. The area overhead including wiring is less than 3% of the total GPU area (815 mm² for Volta V100).

Power. The power consumption of these components is estimated using a synthesized Hardware Description Language (HDL) model evaluated with Synopsys Design Compiler and a 45nm NanGate open cell library. Accel-Watch is used to analyze both dynamic and static power. The estimated 9.1 W power corresponds to less than 2% of the total GPU power.

Time. The warp locality tracker computes the memory pattern within 7 cycles and the warp locality score in 10 cycles. Some operations overlap (PC history unit and sharing history unit), while others execute in parallel (pattern detector and final computation for warp locality score). The CTA locality tracker extracts the CTA memory access pattern in 7 cycles and performs the dot product in 5 cycles, resulting in 12 cycles.

VI. Related Work

We propose a smart locality-aware GPU scheduler, VISTA, that dynamically identifies data locality patterns at both the CTA and warp levels to make intelligent scheduling decisions. VISTA proactively detects inter-CTA locality by assigning CTAs to the SM with the most similar memory behavior. Within each SM, VISTA further identifies inter-warp locality and prioritizes warps that preserve data locality. We have extensively compared VISTA to different locality-aware mechanisms in §V. In this section, we discuss other related work that aims to tackle GPU memory bottleneck: Locality-aware scheduling, data sharing, and smart resource management.

Locality-aware scheduling and data sharing mechanism. Several warp schedulers aim to improve L1D cache performance by prioritizing warps likely to produce cache hits and by limiting the number of active warps to reduce cache thrashing [**?**], [11]–[15], [17]–[21]. Locality-aware CTA schedulers [14], [22], [23], [45] improve data locality by clustering nearby CTAs on the same SM [9], [10] or restricts the number of active CTAs [14], [23] to balance TLP and reduce cache contention. However, many of these approaches rely on compile-time analysis, which may be inaccurate for workloads with irregular or input-dependent memory behavior.

Other techniques improve data reuse by modifying cache behavior through bypassing [46], buffering [47], pinning [48], and inter-core cooperation [1], [3], [25]–[27]. None of them jointly consider locality across both CTAs and warps in a unified, proactive manner. Some research [28] locates the on-chip shared memory in the global memory and uses a unified cache to service both L1D and shared memory accesses. VISTA is orthogonal to such techniques; instead, it proactively schedules CTAs and warps by considering data sharing among non-adjacent CTAs and warps.

Smart resource management. Machine learning (ML) approaches [49], [50] are widely used in different areas, such as CPU resource management. Some use RNNs to model memory access patterns [51], a perceptron-based method for prefetch filtering [52], LSTMs and attention mechanisms to address cache replacement as a sequence prediction problem [38], and reinforcement learning for designing cost-effective cache replacement policies [53]. In the context of CPU-GPU systems, some propose [54] an ML-based prefetching approach for unified virtual memory to mitigate on-demand paging overhead. While ML has gained traction in CPU resource management, its application to GPU resource management remains underexplored.

VII. Conclusion

VISTA, a smart locality-aware GPU scheduler, addresses key memory bottlenecks in memory-intensive GPGPU applications by exploiting data sharing across both adjacent and non-adjacent CTAs and warps. Unlike prior approaches, VISTA dynamically identifies data locality patterns at both the CTA and warp levels and proactively schedules computations to preserve locality. This strategy significantly reduces redundant memory accesses, leading to a 48.1% improvement in performance and a 51.8% reduction in energy consumption compared to the Baseline. These results highlight the potential of VISTA as a lightweight and effective solution for optimizing GPU performance and energy efficiency, making it a promising solution for future memory-intensive applications and high-throughput computing environments.

VIII. Acknowledgment

The authors would like to express their gratitude to the anonymous reviewers for their valuable feedback. This work is part of the project PID2023-146511NB-I00 funded by the Spanish Ministry of Science, Innovation and Universities MICIU /AEI /10.13039/501100011033 and EU ERDF. This publication is promoted by the Barcelona Zettascale Laboratory, backed by the Ministry for Digital Transformation and of Public Services, within the framework of the Recovery, Transformation, and Resilience Plan – funded by the European Union – NextGenerationEU.

REFERENCES

[1] S. Dublish, V. Nagarajan, and N. Topham, "Cooperative caching for GPUs," *ACM TOCS*, 2016.

[2] M. Sadrosadati, A. Mirhosseini, A. Hajiabadi, S. B. Ehsani, H. Falahati, H. Sarbazi-Azad, M. Drumond, B. Falsafi, R. Ausavarungnirun, and O. Mutlu, "Highly concurrent latency-tolerant register files for GPUs," *ACM TOCS*, 2021.

[3] H. Falahati, M. Sadrosadati, Q. Xu, J. Gómez-Luna, B. S. Latibari, H. Jeon, S. Hesaabi, H. Sarbazi-Azad, O. Mutlu, M. Annavaram *et al.*, "Cross-core data sharing for energy-efficient gpus," *ACM TACO*, 2022.

[4] M. Sadrosadati, S. B. Ehsani, H. Falahati, R. Ausavarungnirun, A. Tavakkol, M. Abaee, L. Orosa, Y. Wang, H. Sarbazi-Azad, and O. Mutlu, "ITAP: Idle-time-aware power management for GPU execution units," *ACM TACO*, 2019.

[5] G. Koo, H. Jeon, and M. Annavaram, "Revealing Critical Loads and Hidden Data Locality in GPGPU Applications," in *IISWC*, 2015.

[6] H. Falahati, M. Abdi, A. Baniasadi, and S. Hessabi, "Power-efficient prefetching on GPGPUs," *The Journal of Supercomputing*, 2015.

[7] H. Falahati, M. Abdi, A. Baniasadi, and S. Hessabi, "ISP: Using Idle SMs in Hardware-based Prefetching," in *CADS*, 2013.

[8] T. G. Rogers, M. OConnor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *ISCA*, 2012.

[9] L.-J. Chen, H.-Y. Cheng, P.-H. Wang, and C.-L. Yang, "Improving GPGPU Performance via Cache Locality Aware Thread Block Scheduling," *IEEE CAL*, 2017.

[10] A. Tabbakh, M. Annavaram, and X. Qian, "Power Efficient Sharing-Aware GPU Data Management," in *IPDPS*, 2017.

[11] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N.Patt, "Improving GPU performance via large warps and two-level warp scheduling," in *MICRO*, 2011.

[12] R. Ausavarungnirun, S. Ghose, O. Kayiran, G. H. Loh, C. R. Das, M. T. Kandemir, and O. Mutlu, "Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance," in *PACT*, 2015.

[13] A. Jog, O. Kayiran, A. Pattnaik, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Exploiting core criticality for enhanced GPU performance," in *SIGMETRICS*, 2016.

[14] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither More nor Less: Optimizing Thread-level Parallelism for GPGPUs," in *PACT*, 2013.

[15] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *MICRO*, 2012.

[16] A. Sethia, D. A. Jamshidi, and S. Mahlke, "Mascar: Speeding up GPU Warps by Reducing Memory Pitstops," in *HPCA*, 2015.

[17] A. Jog, O. Kayiran, N. Chidambaram, A. K.Mishra, M. T.Kandemir, O. Mutlu, R. Iyer, and C. R.Das, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," in *ASPLOS*, 2013.

[18] D. Li, M. Rhu, D. R. Johnson, M. O'Connor, M. Erez, D. Burger, D. S. Fussell, and S. W. Keckler, "Priority-Based Cache Allocation in Throughput Processors," in *HPCA*, 2015.

[19] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron, "Energy-efficient Mechanisms for Managing Thread Context in Throughput Processors," in *ISCA*, 2011.

[20] A. Sethia and S. Mahlke, "Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution," in *MICRO*, 2014.

[21] S. Dublish, V. Nagarajan, and N. Topham, "Poise: Balancing thread-level parallelism and memory system performance in GPUs using machine learning," in *HPCA*, 2019.

[22] L. Wang, X. Zhao, D. Kaeli, Z. Wang, and L. Eeckhout, "Intra-cluster coalescing to reduce gpu noc pressure," in *IPDPS*, 2018.

[23] J. Wang, N. Rubin, A. Sidelnik, and S. Yalamanchili, "Laperm: Locality aware scheduler for dynamic parallelism on gpus," in *ISCA*, 2016.

[24] A. Mohammadpur-Fard, S. Darabi, H. Falahati, N. Mahani, and H. Sarbazi-Azad, "Exploiting direct memory operands in gpu instructions," *IEEE CAL*, 2024.

[25] N. Nematollahi, M. Sadrosadati, H. Falahati, M. Barkhordar, and H. Sarbazi-Azad, "Neda: Supporting direct inter-core neighbor data exchange in GPUs," *IEEE CAL*, 2018.

[26] M. M. Keshtegar, H. Falahati, and S. Hessabi, "Cluster-based approach for improving graphics processing unit performance by inter streaming multiprocessors locality," *IET Computers & Digital Techniques*, 2015.

[27] N. Nematollahi, M. Sadrosadati, H. Falahati, M. Barkhordar, M. P. Drumond, H. Sarbazi-Azad, and B. Falsafi, "Efficient nearest-neighbor data sharing in GPUs," *ACM TACO*, 2020.

[28] S. Darabi, E. Yousefzadeh-Asl-Miandoab, N. Akbarzadeh, H. Falahati, P. Lotfi-Kamran, M. Sadrosadati, and H. Sarbazi-Azad, "OSM: Off-chip shared memory for GPUs," *IEEE TPDS*, 2022.

[29] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *ISPASS*, 2009.

[30] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009.

[31] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on GPUs," in *IISWC*, 2012.

[32] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to GPU codes," in *IEEE InPar*, 2012.

[33] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, 2012.

[34] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *IISWC*, 2013.

[35] Xu, Qiumin and Jeon, Hyeran and Annavaram, Murali, "Graph processing on GPUs: Where are the bottlenecks?" in *IISWC*, 2014.

[36] NVIDIA. (2009) Cuda sdk 2.3. [Online]. Available: https://developer.nvidia.com/cuda-toolkit-23-downloads

[37] Baidu. (2017) Deepbench. [Online]. Available: https://github.com/baidu-research/DeepBench

[38] Z. Shi, X. Huang, A. Jain, and C. Lin, "Applying deep learning to the cache replacement problem," in *MICRO*, 2019.

[39] S. Mostofi, H. Falahati, N. Mahani, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Snake: A variable-length chain-based prefetching for gpus," in *MICRO*, 2023.

[40] A. Barnes, F. Shen, and T. G. Rogers, "Mitigating gpu core partitioning performance effects," in *HPCA*, 2023.

[41] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *ISCA*, 2020.

[42] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *MICRO*, 2021.

[43] NVIDIA. (2017) Volta architecture whitepaper. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[44] S. J. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE JSSC*, 1996.

[45] S. Darabi, N. Mahani, H. Baxishi, E. Yousefzadeh-Asl-Miandoab, M. Sadrosadati, and H. Sarbazi-Azad, "NURA: A framework for supporting non-uniform resource accesses in GPUs," *ACM PoMACS*, 2022.

[46] X. Xie, Y. Liang, Y. Wang, G. Sun, and T. Wang, "Coordinated static and dynamic cache bypassing for gpus," in *HPCA*, 2015.

[47] H. Choi, J. Ahn, and W. Sung, "Reducing off-chip memory traffic by selective cache management scheme in gpgpus," in *ACM GPGPUs*, 2012.

[48] G. Koo, Y. Oh, W. W. Ro, and M. Annavaram, "Access pattern-aware cache management for improving data utilization in gpu," in *ISCA*, 2017.

[49] H. Falahati, M. Peyro, H. Amini, M. Taghian, M. Sadrosadati, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Data-Aware compression of neural networks," *IEEE CAL*, 2021.

[50] Sze, Vivienne and Chen, Yu-Hsin and Yang, Tien-Ju and Emer, Joel S, "Efficient processing of deep neural networks," *Synthesis Lectures on Computer Architecture*, 2020.

[51] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," in *PMLR*, 2018.

[52] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *ISCA*, 2019.

[53] S. Sethumurugan, J. Yin, and J. Sartori, "Designing a cost-effective cache replacement policy using machine learning," in *HPCA*, 2021.

[54] X. Long, X. Gong, B. Zhang, and H. Zhou, "Deep learning based data prefetching in cpu-gpu unified virtual memory," *Elsevier JPDC*, 2023.

Dynamic Local Usage: An accurate model for usage of tile-internal wiring in Global Routing

Tilmann Bihler
Research Institute for Discrete Mathematics
University of Bonn
Bonn, Germany
bihler@dm.uni-bonn.de

Daniel Blankenburg
Research Institute for Discrete Mathematics
University of Bonn
Bonn, Germany
blankenburg@dm.uni-bonn.de

***Abstract*—Traditional global routing simplifies routing as a Steiner Tree packing problem on a grid graph, often neglecting the need to account for tile-internal wiring connections to pin shapes. Local wiring usage is typically pre-estimated, leading to inaccuracies. To address this gap, we introduce the Dynamic Local Usage (DLU) model, which evaluates wiring usage based on exact wire shapes and optimizes routes up to detailed pin shapes. Key innovations include tip-to-tip penalty extensions to accurately model packing density and new algorithms to optimize routes according to the DLU congestion model. We demonstrate the superiority of the Dynamic Local Usage model by comparing results after global and detailed routing to an industrial global router on the 3nm and 5nm technology nodes. Our model achieves significant improvements in the number of vias, scenic routes, design rule violations, and timing metrics after detailed routing. Furthermore, the DLU provides a better support for incremental design changes, such as pin movements, by dynamically updating local wiring usage during routing.**

***Index Terms*—Physical Design, VLSI, Placement and routing**

I. Introduction

Routing in VLSI design is tasked with the computation of rectilinear Steiner trees that have to obey numerous constraints and design rules. In practice different objectives should be optimized and timing constraints need to be met. Due to its high complexity the routing process is traditionally split into at least two steps: *global routing* and *detailed routing*. In global routing the task is simplified by introducing a rectangular tiling that covers the chip area. Tile centers of adjacent tiles are connected, resulting in a grid graph. We assume that the chip area consists of L layers, each having a preferred dimension (x or y) in which wires must be routed. Such a grid graph is depicted in Fig. 1.

Traditionally, all pre-existing wires (prewires) and pins are projected onto the centers of these tiles, and Steiner trees are constructed on the global routing grid graph. This approach formulates the global routing problem as a Steiner tree packing problem, which often overlooks the necessity of connecting to the exact pin shapes. Consequently, the usage caused by tile-internal wiring is typically estimated prior to global routing and remains unadjusted based on the actual global routing solution [1]–[3]. While there is an extensive body of work dedicated to developing algorithms for solving the global routing problem (see surveys [4], [5]), relatively few studies focus on modeling capacity and wiring usage accurately. An improved model for local wiring usage offers several significant advantages:

- *Enhanced Input for Detailed Routing*: A more precise usage model provides the detailed router with better guidance, leading to higher-quality routing solutions.
- *Support for Incremental Design Changes*. During timing optimization based on global wires, small adjustments—such as slight pin movements or buffer insertions—are often made to an already globally routed design. These changes predominantly affect tile-internal wiring usage and may go unnoticed if local usages are estimated only once prior to routing.
- *Improved Scalability*: By accurately modeling tile-internal wiring usage, larger global routing tile sizes can be utilized without significantly degrading solution quality [6].

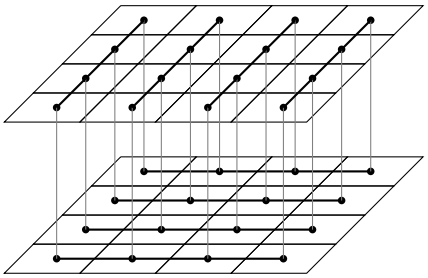


Fig. 1: A global routing grid graph with $L = 2$ layers

A. Our contribution

In this work, we introduce the Dynamic Local Usage model, a novel approach that dynamically adjusts local wiring usage based on the actual routing solution. Our key contributions are:

1) *An Intuitive Area-Based Capacity and Usage Model*: We present a capacity and usage model that applies to all wires and vias, enhanced by tip-to-tip penalty extensions. These extensions accurately model the packing inefficiencies inherent in detailed routing, leading to a more precise congestion assessment.
2) *A set of novel RC-aware algorithms on exact wire shapes*: These algorithms incorporate the tip-to-tip penalty extensions and improve delays on the global routes significantly.
3) *Superior routing results*: Through extensive experimentation (Table II, Table I), we demonstrate that our DLU model outperforms a state-of-the-art industrial global router in terms of key metrics after detailed routing, including wirelength, via count, detour factors (scenics), violation counts, and timing metrics.

Given the significant benefits of the Dynamic Local Usage model, DLU has been adopted as the default setting in our industrial VLSI routing toolkit, highlighting its practical impact and effectiveness in modern design environments.

II. The dynamic local usage model

A. Area usages

a) Edge capacities: Let $e = \{u, v\}$ be an edge in the global routing graph. The edge area of e, denoted as $\texttt{area}(e)$, is the rectangle spanned by u and v, extending to the tile borders in the

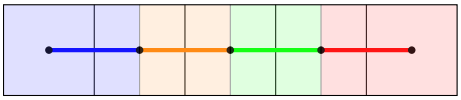


Fig. 2: Global routing edges e,f,g,h in a tile corridor and their corresponding edge areas

non-preferred direction, and further to the chip border if u or v are boundary nodes (Fig. 2). The capacity of e is determined as the total unblocked default track length available in $\texttt{area}(e)$ multiplied with the default wiring pitch $\texttt{p}_z$ (width plus spacing of the default wiretype) on the layer z of edge e.

$$\texttt{cap}(e) = \texttt{unblocked_track_length}(\texttt{area}(e)) \cdot \texttt{p}_z \quad (1)$$

To compute the unblocked default track length, the exact positions and spacing requirements of all fixed objects (pins, blockages, and locked input wires) within $\texttt{area}(e)$ are considered, and the blocked track length is subtracted from the total default track length in $\texttt{area}(e)$. When tracks are tightly packed and no fixed objects are present, $\texttt{cap}(e)$ equals the total area of $\texttt{area}(e)$.

b) Area usage of a Steiner tree: We proceed to define the area usage of a Steiner tree Y from edge e. Let $w \in E(Y)$ be a wire or via. Let us denote the rectangular shape of the wire or viapad by $\texttt{shape}(w)$. It would be natural to compute the area usage based on the union of shapes of Y, however this is too optimistic. We define a *unit space usage*, denoted by $\texttt{unit_usage}(w)$ (see Section II-B) for every wiretype and viapad on every layer. Let $\texttt{usage_shape}(w)$ be the rectangle that has the same centerline as $\texttt{shape}(w)$ in preferred dimension and a width of $\texttt{unit_usage}(w)$ in non-preferred dimension. Again, simply computing the usage on the area of the usage shapes would be too optimistic, due to the heavy simplification of design rules in global routing. Instead, we must account for the inevitable increases in vias (10–20%) and wirelength (5%) during detailed routing (see Table II, Table I). With Dynamic Local Usage, we take a precise approach by introducing *tip-to-tip penalty extensions*, which account for packing inefficiencies and wire spreading during the transition to detailed routing. These tip-to-tip penalty extension (see Section II-C) are applied to wire ends and via shapes in preferred direction, see Fig. 5. Let $\texttt{t2t_usage_shape}(w)$ be $\texttt{usage_shape}(w)$ extended by the tip-to-tip penalty extension (defined in Section II-C) in positive and negative preferred direction.

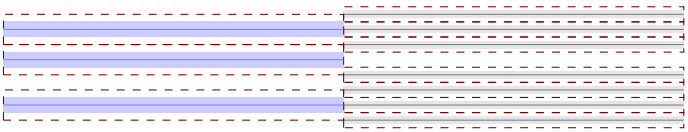
Finally, we define the usage of Y from e as the intersection area of the union of all the tip-to-tip extended usage shapes of Y with $\texttt{area}(e)$, divided by the capacity of e.

$$\texttt{usg}_Y(e) := \frac{|\texttt{area}(e) \cap \bigcup_{w \in E(Y)} \texttt{t2t_usage_shape}(w)|}{\texttt{cap}(e)}. \quad (2)$$

B. Non-default wire usage

The capacity of an edge is measured as the available default track length times the default wiring pitch. Naturally, the unit space usage of a default wire w on layer z is given by the default pitch on layer z. However, non-default wiretypes, which are not necessarily placed on the default track pattern, require special consideration. A naive approach is to measure the usage by comparing the wire pitch of the non-default wiretype and the default wiretype. This neglects that in many cases, non-default track patterns can be defined in a more sophisticated way such that they have fewer conflicts with the default track pattern.

When specific track patterns for non-default wiretypes are available (as in our experiments), we compute their unit space usage by taking the average between the worst-case and average-case scenarios regarding the number of default tracks they block. This is illustrated in Fig. 3a. Similarly, the unit space usages of via pad shapes are computed by considering the number of blocked default tracks. As described in Section II-A, we compute usage shapes based on the unit space usages, as illustrated in Fig. 3b.



(a) Track patterns for default (gray) and double-wide (blue) wiretypes are shown, with spacing constraints in red. In the worst case, a blue wire blocks three default tracks, while on average, it blocks $\frac{7}{3}$ tracks. This gives a unit space usage of $\texttt{unit_usage}(w) = 0.5 \times (3 + \frac{7}{3})\texttt{p}_z = \frac{8}{3}\texttt{p}_z$.



(b) Transformation of the shape of w into its usage shape.

Fig. 3: Illustration of the usage shape computation for non-default wiretypes.

A detailed overview on estimation of wire usages based on track patterns can be found in [7].

C. Tip-to-tip penalty extensions

Tip-to-tip penalty extensions model two key effects observed during detailed routing:

- *Packing inefficiencies*: Gaps larger than the minimum tip-to-tip spacing are left between wires due to the complexity of detailed routing.
- *Spreading of wires*: Overlapping wires after global routing must be spread to different tracks in detailed routing, leading to elongation of adjacent wires on neighboring layers and increased space usage near route bends.

To address these effects, we extend $\texttt{usage_shape}(w)$ of a wire w with wiretype t on layer z on both ends by some length $\texttt{penalty}_{z,t} > 0$ and calculate the area usage of the resulting extended shape $\texttt{t2t_usage_shape}(w)$ as discussed in Section II-A. We proceed analogously for via pad shapes and stacked vias. If a via pad shape has a larger unit space usage than the corresponding wire shape (a so-called *clubfoot via*), we extend the via pad shape in both positive and negative preferred direction by $\texttt{penalty}_{z,t}$, as shown in Fig. 5. Fig. 4 shows the planar projection of a global route with tip-to-tip penalty extensions. The penalty length is determined by:

$$\texttt{penalty}_{z,t} := \min \left\{ \begin{array}{l} \frac{k_{\text{tip}}}{2} \cdot \texttt{t2t}_{z,t}, \\ \frac{k_{\text{tip}}}{2} \cdot 0.6 \cdot (\texttt{p}_{z+1,t} + \texttt{p}_{z-1,t}), \\ 0.4 \cdot \texttt{length}_{tile} \end{array} \right\}, \quad (3)$$

where

- $\texttt{t2t}_{z,t}$ is the minimum required spacing in preferred dimension of two wires with wiretype t (*tip-to-tip spacing*) on layer z,
- $\texttt{p}_{z,t}$ is the *pitch*, i.e. the minimum required distance between two tracks, for wiretype t on layer z,
- $\texttt{length}_{tile}$ denotes the average length of a global routing tile,
- $k_{tip} \geq 1$ is a tunable penalty parameter dependent on tile size, technology node, and the detailed router's packing efficiency.

We discuss our choice of parameters in Section IV. (3) incorporates the following:

- *Minimum required tip-to-tip spacing*: Note that $\frac{\texttt{t2t}_{z,t}}{2}$ is the minimum required area usage that should be charged at the

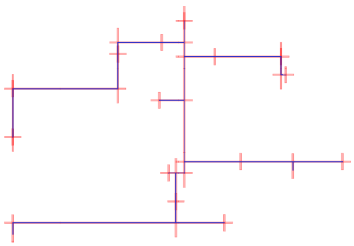


Fig. 4: A global route with tip-to-tip penalty extensions

end of a wire. Indeed, if two wires are abutting, they require a minimum spacing of $\texttt{t2t}_{z,t}$ between them, and it is natural to account for half of this space usage at each of the wires.

- *Pitch of adjacent layers*: These affect the spreading on adjacent layers and thus the potential extension of the wire segment on layer z during detailed routing.
- *Tilesize bound*: The last term prevents overcongestion on higher layers where tip-to-tip spacing becomes large due to thicker wires.



Fig. 5: Illustration of the union of a wire usage shape with the usage shape of a clubfoot via pad (blue) with tip-to-tip penalty extensions (red) applied at both ends of the usage shapes. The area usage of the wire and the clubfoot via is computed according to the union of the grey and the red rectangles.

III. Algorithms

A. The global routing algorithm

We formulate the fractional relaxation of the global routing problem as a resource sharing problem and solve it approximately with the algorithm described in [8]. The resource sharing algorithm proceeds in several phases and computes multiple trees for each net. With DLU, also the local wiring is optimized and recomputed in every tree computation. After that we apply randomized rounding to obtain an integral solution and conclude with a post-optimization, that is similar to typical ripup & reroute approaches. Our entire algorithmic framework is timing-aware. The resource sharing formulation includes timing constraints as described in [9] and our Steiner tree approximation algorithm follows a similar approach to [10]. Let $\mathcal{R}$ denote the set of resources, including edges in the global routing graph, netlength resources, and timing resources for every sink in each net. To ensure convergence to an optimal solution, the Steiner tree approximation algorithm has to find a tree Y that minimizes the total cost:

$$\texttt{cost}(Y) = \sum_{r \in \mathcal{R}} \texttt{price}(r)\texttt{usg}_Y(r) \tag{4}$$

As in most global routing algorithms the price of resource r, $\texttt{price}(r)$, is exponential in the load of r. The usage $\texttt{usg}_Y(r)$ represents the area usage for edge resources (as discussed in Section II-A) and the RC-delay for sink resources within tree Y.

Within the dynamic local usage model the tree Y on which we measure the usages is connected to the exact pin shapes. Our algorithm to find such a tree proceeds in three steps:

1) RC-aware Steiner tree algorithm on the global routing graph (Section III-B),
2) RC-aware optimization of x- and y-coordinates within tiles (Section III-C),
3) RC-aware layer embedding algorithm (Section III-D).

B. Finding tile-center to tile-center routes

We employ a Steiner tree approximation algorithm with a Prim-like path decomposition heuristic on the three-dimensional global routing grid graph. To handle RC-delays, we adopt techniques similar to those in [9], [10], which involve splitting certain components and reconnecting them to the source pin via RC-aware path searches

The inclusion of tip-to-tip penalty extensions introduces additional costs at vias in the route, making the cost dependent on the route's structure. This complexity means that considering only edge costs in the path search algorithm is insufficient. To address this, we model the problem as a Steiner tree problem with node costs. More precisely, for a node v with incident preferred-dimension edges e and f, the node cost is the sum of the tip-to-tip penalty extension costs on e and f.

We adjust the edge costs of a preferred-dimension edge e by subtracting twice the tip-to-tip penalty cost from its original cost, ensuring the result is non-negative by taking the maximum with zero. For via edges, we charge congestion costs only for the tip-to-tip penalties caused by overhanging via pad shapes. Other cost components, such as those for timing or netlength resources, remain unchanged. Given that the maximum degree of a node in the global routing graph is 4 and assuming that the tip to tip penalty costs are small compared to the edge costs, this can be shown to yield a close-to 3-approximation for this Steiner tree problem with node costs, as shown in [6].

C. RC-aware XY-optimization

Given the tile-center to tile-center route, we first realise tile-internal connections to the pin shapes with a rectilinear minimum Steiner tree routine.

We then optimize the x- and y-positions of the Steiner nodes within their tiles by considering the two-dimensional projection of the route. Our algorithm employs an improved version of the algorithm from [11] and incorporates:

- *Congestions costs*: In the presence of area usages, tile-internal movement of nodes affects the congestion costs. To account for this in the planar case, we consider the worst congestion over the assigned layers which follow the preferred dimension of a segment.
- *RC-Delays*: We include RC-delays heuristically by estimating the downstream capacitance of a node by assuming the minimum capacitance over all layers for all segments in the subtree.

The algorithm optimizes x- and y-coordinates independently and proceeds as a dynamic program in reverse topological order on Y, propagating piecewise linear functions for the subtree costs. For a tree Y our algorithm has running time $\mathcal{O}(|V(Y)| \cdot m)$, where m is the maximum number of breakpoints occuring in the propagation of the piecewise linear functions. In the worst case the number of breakpoints can be linear in $|V(Y)|$, but has been shown to be much smaller in practice experimentally [6].

D. RC-aware layer embedding

We assume that the input to the layer embedding algorithm is a source-rooted planar arborescence, possibly containing tile-internal diagonal arcs between nodes. Our layer embedding supports these

Input: A source-rooted planar arborescence Y.
Output: A layer embedding of $E(Y)$.

1 **for** *each edge* $e = (u, v)$ *in* Y *in reverse topological order* **do**
2 Let $\delta_Y^+(v)$ be the set of outgoing edges from v.
3 **for** *each ordering* $e_1, e_2, \ldots, e_k$ *of* $\delta_Y^+(v)$ **do**
4 // Fixing the ordering allows us to approximate RC-delay costs of connecting via edges
5 **for** *each possible layer* $l \in \{1, ..., L\}$ **do**
6 Compute the minimal cost of assigning layer l to edge e, considering:
7 - The minimal costs of child edges e_i, assigned to layers compatible with l, following the fixed ordering.
8 - Congestion and RC-delay costs of edge e and connecting via edges, based on approximated downstream capacitance from child edges.
9 Store the minimal cost and corresponding layer assignments of the child edges.
  end
  end
end
10 **return** the minimal cost assignment at the root edge.

Algorithm 1: The layer embedding dynamic program

diagonal arcs by selecting the cheaper of the two possible L-shaped realizations for each. We split segments at edge area borders to ensure we find an optimal congestion-aware solution.

Dynamic programming is a standard technique for layer embedding, particularly in global routing methods that separate 2D routing from layer assignment [12], [13]. The layer embedding algorithm of [14] also considers RC-delays and congestion costs in a dynamic program, albeit without stating a runtime or bounds on the solution quality.

Similar to these approaches, our algorithm traverses the tree in reverse topological order, embedding segments while considering the optimal embeddings of child arcs in the subtree.

Key distinctions of our algorithm include:

- Embedding exact wire shapes and considering tip-to-tip penalty extensions.
- *Fast estimates for RC-costs*: We estimate downstream capacitance by assuming minimum wire capacitances for all wires in the subtree, and using a correction factor based on a lower bound on upstream resistance, similar to the method for RC-aware path search in [10].

Incorporating RC costs adds complexity because the order in which child branches are embedded affects the delay costs of vias.

Our algorithm achieves optimal embeddings in the timing-unaware setting and provides a good a posteriori approximation ratio in the RC-aware setting. It is described in pseudo-code as Algorithm 1.

Since nodes in the planar tree have a bounded degree, we can feasibly explore all possible orderings of child embeddings, that is, the for-loop in Line 3 of Algorithm 1 has a constant number of iterations. It is an improved version of the algorithm described in [15], using a very similar definition of RC-aware costs, improving on the running time by a factor of L. The key insight that leads to the running time improvement is that for a fixed ordering, the minimal cost assignments can be computed recursively bottom-up and top-down for all layers, leading to an accumulated runtime of $\mathcal{O}(L)$ for the for-loop in Line 5 of Algorithm 1. Details can be found in [6].

Theorem 1: Algorithm 1 computes a $(1 + (\alpha - 1)(1 - \beta))$-approximately optimal solution within a running time of $\mathcal{O}(|E(Y)| \cdot L)$. Here $\alpha \leq c_{\max}/c_{\min}$ is an upper bound on our downstream capacitance estimate error and $\beta \geq r_{\min}/r_{\max}$ is a lower bound on our upstream resistance estimate error. The values $c_{\min}$, $c_{\max}$, $r_{\min}, r_{\max}$ denote the minimum and maximum wire capacitance, respective resistance across all layers.

Note that α is an upper bound on the approximation ratio and $\alpha \leq c_{\max}/c_{\min}$ holds. This ratio is close to 1 in recent technologies. Details can be found in [6].

IV. Results

A. Test setup

All experiments were conducted on a 2×64 AMD EPYC 7742 CPU machine with 2TB of RAM, utilizing 32 threads. The Dynamic Local Usage model was implemented within an industrial VLSI design toolkit. For comparison, results were benchmarked against a traditional global router from the same toolkit by running the same industrial detailed router on the global solutions produced by both models.

The traditional global router is also RC-aware and follows a similar methodology to the one described in Section III, albeit employing simpler pin connection algorithms.

The timing-unaware experiments were performed on a combined testbed of 95 state-of-the-art microprocessor units from the 3nm and 5nm technology nodes. Meanwhile, the timing-aware experiments were done on a testbed of 17 state-of-the-art microprocessor units from the 5nm technology node. All test designs were extracted from an industrial design environment prior to routing.

For the experiments using the DLU model, a tip-to-tip penalty factor $k_{tip} = 14.5$ was set for the 5nm designs and $k_{tip} = 13$ for the 3nm designs. All runs were conducted with a global routing tilesize of about 70 times the minimum wiring pitch across all layers.

B. Experimental results in a timing-unaware routing flow

a) Metrics: To evaluate the efficacy of the DLU model in guiding detailed routing, we compared its performance to that of the traditional global router (trad) across several metrics on the mixed testbed of 95 designs from the 3nm and 5nm technology nodes. The measured metrics in Table II include:

- *Wirelength and Vias*: Total wirelength and the number of vias after global routing (GR WL, GR Vias) and detailed routing (DR WL, DR Vias).
- *Global Routing Runtime*: The total runtime of global routing (GR Walltime).
- *Violations*: The total number of design rule violations (DRVs) as well as the number of wires and vias causing one or more spacing violations (Violation Wires, Violation Vias).
- *Scenics*: The number of nets that were routed 25%, 50%, and 100% longer than a shortest rectilinear Steiner tree after detailed routing (DR Scenics 25, 50, 100).

b) Results and Discussion: Table II demonstrates that the DLU model shows significant improvements over the traditional global router:

- *Wirelength and Vias*: A substantial reduction in the number of vias and a wirelength were observed after global routing. These improvements stem from the more accurate congestion model and the optimized local wiring strategies discussed in Section III. The traditional router also computes connections to the local pin

Chip nets	Run	GR WL (m)	GR Vias	wACE4	GR TnS (ps)	GR WS (ps)	GR Walltime	DR WL (m)	DR Vias	DRVs	Violation Wires	Violation Vias	DR WS (ps)	DR TnS (ps)
U_1	trad	0.0402	57651	85.99 %	-7762	-73	00:00:21	0.0432	58557	0	0	0	-75	-8529
7096	DLU	0.0403	54437	86.75 %	-7278	-71	00:00:32	0.0432	58238	3	1	1	-76	-8637
U_2	trad	6.4131	3339916	90.08 %	-105472	-147	00:09:17	6.6953	3469906	175	3	2	-164	-233517
325295	DLU	6.3173	3178308	90.27 %	-75443	-112	00:17:42	6.6010	3479066	187	0	0	-118	-203004
U_3	trad	12.2375	8126263	88.21 %	-365071	-84	00:22:16	12.9010	8538964	248	30	28	-100	-708008
815646	DLU	12.1876	7561152	90.24 %	-258110	-84	00:32:41	12.8604	8574487	170	2	5	-85	-703788
U_4	trad	2.1136	1475502	88.37 %	-28233	-48	00:05:27	2.1930	1537545	1155	0	0	-67	-72586
188249	DLU	2.1086	1432395	82.75 %	-22240	-43	00:07:32	2.1905	1537156	1147	0	0	-68	-70606
U_5	trad	0.7596	545084	79.80 %	-18	-4	00:02:06	0.7887	558179	179	0	0	-18	-173
63620	DLU	0.7578	518403	89.28 %	-6	-5	00:05:26	0.7874	555995	179	0	0	-16	-134
U_6	trad	0.0760	83973	74.74 %	-19386	-162	00:00:24	0.0797	84595	74	0	0	-181	-19890
11264	DLU	0.0758	77183	73.92 %	-19307	-162	00:00:31	0.0796	83232	79	0	0	-177	-19802
U_7	trad	1.9506	1797534	88.81 %	-31950	-35	00:04:39	2.0744	1874937	53	0	27	-56	-83968
193659	DLU	1.9350	1683718	88.73 %	-24626	-31	00:06:23	2.0629	1881236	31	1	19	-55	-77767
U_8	trad	1.1089	2166946	87.50 %	-18724	-33	00:05:12	1.2048	2025030	16	0	0	-51	-51659
247022	DLU	1.1080	1787859	86.12 %	-12191	-33	00:08:49	1.2037	2005529	18	0	0	-54	-51356
U_9	trad	1.0524	2017272	87.31 %	-6916	-25	00:05:05	1.1434	1898388	21	0	0	-49	-34706
228521	DLU	1.0524	1665140	86.01 %	-4873	-24	00:08:29	1.1440	1888193	26	0	0	-49	-33219
U_{10}	trad	0.8398	676946	82.55 %	-5901	-35	00:02:21	0.8779	693251	98	1	4	-49	-12467
84125	DLU	0.8374	652625	83.90 %	-5050	-35	00:04:05	0.8765	693337	82	0	0	-50	-11785
U_{11}	trad	0.8707	862294	84.58 %	-4638	-24	00:01:59	0.9264	888493	19	0	0	-25	-5836
89263	DLU	0.8698	799497	85.61 %	-4530	-24	00:03:11	0.9251	882951	18	0	0	-24	-6153
U_{12}	trad	0.0570	61944	81.01 %	-14698	-549	00:00:20	0.0603	62356	29	0	0	-557	-15606
9250	DLU	0.0570	59695	81.13 %	-14346	-538	00:00:25	0.0602	62277	30	0	0	-557	-15564
U_{13}	trad	0.0007	882	55.81 %	0	46	00:00:11	0.0007	890	0	0	0	44	0
149	DLU	0.0007	842	48.65 %	0	44	00:00:11	0.0007	850	0	0	0	42	0
U_{14}	trad	0.4926	526766	86.54 %	-214177	-189	00:01:30	0.5212	562857	9	0	0	-197	-224122
68817	DLU	0.4902	487210	85.25 %	-210666	-189	00:02:32	0.5198	560112	16	0	0	-197	-224807
U_{15}	trad	0.0841	108281	83.33 %	-20841	-324	00:00:35	0.0893	112001	14	0	0	-326	-23033
14286	DLU	0.0834	103462	82.69 %	-20614	-322	00:00:54	0.0886	111373	14	0	0	-325	-23269
U_{16}	trad	0.4672	550596	86.75 %	-28736	-90	00:01:21	0.4963	562434	7	0	0	-98	-61724
67837	DLU	0.4655	511831	86.23 %	-20504	-86	00:02:01	0.4950	563610	9	0	0	-96	-63673
U_{17}	trad	1.1403	1196247	83.76 %	-21472	-93	00:02:53	1.1992	1220630	21	0	0	-47	-41588
144203	DLU	1.1359	1114358	83.83 %	-10116	-63	00:05:13	1.1966	1218459	30	0	0	-48	-45375
Summary	trad	29.7043	23594097		-893993		01:05:57	31.2949	24149013	2118	34	61		-1597411
	DLU	29.5227	21688115		-709900		01:46:37	31.1353	24156101	2039	4	25		-1558941
Δ%		-0.61 %	-8.08 %		-20.59 %		+61.66 %	-0.51 %	+0.03 %	-4 %	-88.24 %	-59.02 %		-2.41 %

TABLE I: Results in a timing-aware routing flow after global and detailed routing

Metric	Traditional	Dynamic Local Usage	
GR Wire length	505.7703m m	477.7347 m	-4.84 %
DR Wire length	498.2275 m	497.2327 m	-0.20 %
GR Vias	288584789	252005360	-12.19 %
DR Vias	301822488	300595525	-0.41 %
GR Walltime (hh:mm:ss)	08:23:29	10:13:48	+24.12 %
DRVs	13555	12448	-8.17 %
Violation Wires	1418	1293	-8.82 %
Violation Vias	3593	2920	-18.73 %
DR Scenics 25	107229	60651	-43.44 %
DR Scenics 50	17610	7126	-59.53 %
DR Scenics 100	497	473	-4.83 %

TABLE II: Combined results from a timing-unaware routing flow on a testbed of 95 chips (default)

shapes, though in a very naive way. The reductions persisted into detailed routing, though with a smaller magnitude.

- *Violations*: We observe significant improvements in terms of violations, especially the number of violation vias decreases. This highlights the DLU model's capacity to provide a more better input to detailed routing.
- *Scenics*: The sharp decline in the number of DR Scenics indicates fewer routing detours, as the detailed router benefited from an improved global solution.

However, the DLU model incurred a minor overhead in global routing runtime, due to the optimization of several tile-internal nets and the additional xy-optimization and layer embedding algorithms. Note that also tile-internal nets are recomputed several times across different phases in the resource sharing algorithm with DLU. This overhead is relatively minor when compared to the detailed routing runtime, which exceeded 50 hours for this testbed.

C. Experimental results in a timing-aware routing flow

We provide results after global routing and after detailed routing in a timing-aware flow in Table I.

a) Metrics: For the timing-aware experiments, we measured a subset of the metrics from Section IV-B, along with additional timing and congestion metrics:

- *Congestion (wACE4)*: A weighted average of the 5% most congested global routing edges, commonly used to assess routability [16]. While the wACE4 values from the two models are not directly comparable due to differences in usage and congestion calculations, they were similar in magnitude.
- *Timing Metrics*: Worst slack after global and detailed routing (GR WS, DR WS) and the total negative slack (GR TnS, DR TnS). Slacks were computed using an industrial signoff tool using RICE [17].

b) Results and Discussion: The timing-aware experiments yielded results consistent with the timing-unaware testbed, with additional timing-specific insights:

- *Wirelength, Vias and Violations* As in the timing-unaware flow, DLU achieved significant reductions in vias and wirelength. We point out that the global wirelength reduction is not as drastic as in Section IV-B, because the traditional global router optimizes the x- and y-positions of the pin connected routes to improve timing. Improvements in violations were particularly pronounced on unit U_3, where the traditional router underestimated congestion, leading to poor detailed routing performance.
- *Timing Metrics*: Improvements in the FOM after global routing were substantial, driven by RC-aware algorithms described in Section III. Although the detailed router is timing-unaware, heuristic adherence to the global routing solution led to modest improvements in timing metrics post-detailed routing.

Despite the DLU model incurring a larger runtime increase in the timing-aware flow—attributed to the higher complexity of RC-aware optimizations—the overhead is justified by significant reductions in vias and improvements in slack metrics.

V. Conclusion

The Dynamic Local Usage (DLU) model enhances traditional grid-based global routing approaches by dynamically optimizing local wiring usage with exact wire shapes. Incorporating innovative features such as tip-to-tip penalty extensions and novel RC-aware algorithms to optimize local wiring with regard to the DLU model — including a Steiner tree approximation on the global routing graph, planar tile-internal optimization, and a layer embedding algorithm — DLU provides a more accurate input to detailed routing. Our experimental results demonstrate that DLU significantly improves the quality of detailed routing solutions across a variety of metrics, including via count, scenic routes, design rule violations, and timing metrics, while maintaining reasonable computational overhead. The adoption of DLU in our industrial toolkit highlights its practical impact.

References

[1] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, and J. Vygen, "Bonnroute: Algorithms and data structures for fast and good vlsi routing," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, pp. 1–24, 2013.

[2] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy, A. D. Huber, G. E. Tellez, D. Keller, and S. S. Sapatnekar, "Glare: Global and local wiring aware routability evaluation," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 768–773.

[3] J. He, U. Agarwal, Y. Yang, R. Manohar, and K. Pingali, "Sproute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 586–591.

[4] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," *Integration*, vol. 31, no. 1, pp. 1–49, 2001.

[5] H. Tang, G. Liu, X. Chen, and N. Xiong, "A survey on steiner tree construction and global routing for vlsi design," *IEEE Access*, vol. 8, pp. 68 593–68 622, 2020.

[6] T. Bihler, "Dynamic local usage in bonnrouteglobal," Ph.D. dissertation, University of Bonn, 2023.

[7] P. Saccardi, "Continuous routing," Ph.D. dissertation, University of Bonn, 2022.

[8] D. Müller, K. Radke, and J. Vygen, "Faster min–max resource sharing in theory and practice," *Mathematical Programming Computation*, vol. 3, no. 1, pp. 1–35, 2011.

[9] S. Held, D. Müller, D. Rotter, R. Scheifele, V. Traub, and J. Vygen, "Global routing with timing constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 406–419, 2017.

[10] R. Scheifele, "Rc-aware global routing," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–8.

[11] A. K. Kiefner, "Minimizing path lengths in rectilinear steiner minimum trees with fixed topology," *Operations Research Letters*, vol. 44, no. 6, pp. 835–838, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167637716301481

[12] M. Pan, Y. Xu, Y. Zhang, and C. Chu, "Fastroute: An efficient and high-quality global router," *VLSI Design*, vol. 2012, no. 1, p. 608362, 2012.

[13] D. Shi, E. Tashjian, and A. Davoodi, "Dynamic planning of local congestion from varying-size vias for global routing layer assignment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 8, pp. 1301–1312, 2017.

[14] G. Liu, X. Zhang, W. Guo, X. Huang, W.-H. Liu, K.-Y. Chao, and T.-C. Wang, "Timing-aware layer assignment for advanced process technologies considering via pillars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 6, pp. 1957–1970, 2022.

[15] R. Scheifele, "Timing-constrained global routing with rc-aware steiner trees and routing based optimization," Ph.D. dissertation, University of Bonn, 2019.

[16] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy, A. D. Huber, G. E. Tellez, D. Keller, and S. S. Sapatnekar, "Techniques for scalable and effective routability evaluation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 2, Mar. 2014. [Online]. Available: https://doi.org/10.1145/2566663

[17] C. L. Ratzlaff and L. T. Pillage, "Rice: Rapid interconnect circuit evaluation using awe," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 6, pp. 763–776, 1994.

CND-IDS: Continual Novelty Detection for Intrusion Detection Systems

Sean Fuhrman[1], Onat Gungor[2], and Tajana Rosing[2]

[1]Department of Electrical and Computer Engineering, University of California, San Diego
[2]Department of Computer Science and Engineering, University of California, San Diego
{*stfuhrma, ogungor, tajana*}*@ucsd.edu*

***Abstract*—Intrusion detection systems (IDS) play a crucial role in IoT and network security by monitoring system data and alerting to suspicious activities. Machine learning (ML) has emerged as a promising solution for IDS, offering highly accurate intrusion detection. However, ML-IDS solutions often overlook two critical aspects needed to build reliable systems: continually changing data streams and a lack of attack labels. Streaming network traffic and associated cyber attacks are continually changing, which can degrade the performance of deployed ML models. Labeling attack data, such as zero-day attacks, in real-world intrusion scenarios may not be feasible, making the use of ML solutions that do not rely on attack labels necessary. To address both these challenges, we propose CND-IDS, a continual novelty detection IDS framework which consists of (i) a learning-based feature extractor that continuously updates new feature representations of the system data, and (ii) a novelty detector that identifies new cyber attacks by leveraging principal component analysis (PCA) reconstruction. Our results on realistic intrusion datasets show that CND-IDS achieves up to 6.1× F-score improvement, and up to 6.5× improved forward transfer over the SOTA unsupervised continual learning algorithm. Our code is available at https://github.com/Sean-Fuhrman/CND-IDS.**

I. Introduction

In today's digital landscape, cybersecurity is essential for safeguarding sensitive data and maintaining trust in digital systems. With cyber threats becoming more sophisticated, organizations must adopt thorough security measures to protect against breaches and unauthorized access [1]. A critical part of such security measures are intrusion detection systems (IDS) which identify malicious network traffic and computer usage [2]. Recently, Machine Learning (ML) has become a popular IDS solution due to its great attack detection performance and less requirement for human knowledge [3]. However, state-of-the-art (SOTA) ML-IDS solutions often fail to adequately address the challenges posed by (i) the evolving nature of cyber attacks and (ii) the lack of attack labels.

A University of Maryland survey predicts that a new cyber attack happens somewhere on the Internet every 39 seconds [4], underscoring the need for dynamic IDS. However, most SOTA ML-IDS approaches are static (offline) and cannot easily adapt their behavior over time [5]. ML models, when taught new knowledge, *catastrophically forget* previously known information. To efficiently address this problem, continual learning (CL) is proposed which is capable of learning from an infinite stream of data [6]. An experience is defined as a shift in the data stream distribution, and CL models are designed to function well across multiple experiences. This capability is currently essential in cybersecurity, where each new attack type can be considered as a new experience [7]. To illustrate, an intrusion detection model trained for a group of attacks (experiences) such as *worm*, *shellcode*, and *exploits* should be able to identify new types of attacks (experiences) such as *backdoor* and *fuzzers* [8]. Some studies have adopted CL for intrusion detection [7]–[12], but they rely on labeled attack data, which may not always be feasible to obtain. This causes the ML model to overfit to common attacks in the dataset, resulting in a failure to detect rare and zero-day threats.

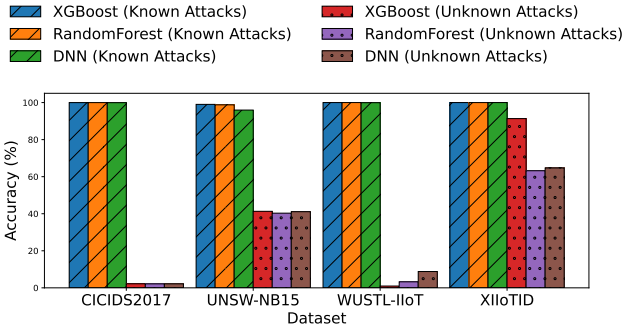


Fig. 1: State of the art machine learning intrusion detection performance on known/unknown attacks

SOTA ML-IDS solutions have been shown to have high accuracy on well represented attacks within the data, however have low precision on infrequent attacks [13]. It is extremely difficult for zero-day attacks to be well represented in the data. Figure 1 illustrates the limitations of current ML-IDS methods in handling zero-day attacks, revealing a substantial decline in performance when faced with previously unseen threats. To address this issue, novelty detection (ND)-based ML methods have demonstrated significant promise in detecting both known and zero-day attacks without requiring a labeled dataset [14]. These ND solutions can effectively identify attacks by analyzing purely normal data, provided that the attacks exhibit substantial deviations from normal behavior [14]. However, it has been shown that ND algorithms such as Isolation Forest and Local Outlier Factor result in lower prediction

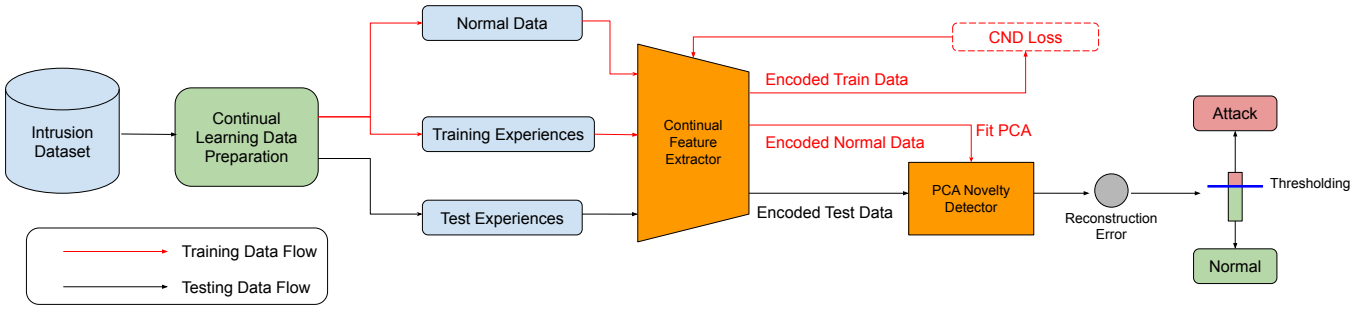


Fig. 2: Continual Novelty Detection IDS (CND-IDS) Framework

performance under changing environments, where they suffer from issues like *catastrophic forgetting* [15]. This issue can be addressed by leveraging continual learning strategies, hence motivating continual novelty detection methods for IDS.

To address the challenges of evolving data streams and lack of supervision, we propose *CND-IDS*, a continual novelty detection approach for ML-based intrusion detection (Fig. 2). To the best of our knowledge, we are the first to propose continual ND for intrusion detection. CND-IDS consists of two critical components: a continual feature extractor (CFE) and a novelty detector based on principal component analysis (PCA) reconstruction. We design a novel continual ND loss to learn representations of the data stream, enabling the CFE to update its representations in a streaming environment with zero attack labels. PCA reconstruction error is then used to classify test samples as either attack or normal data based on thresholding. By fitting PCA on normal data, CND-IDS is trained without any labeled attack data, enabling it to detect previously unknown zero-day attacks. Our results show that on realistic intrusion datasets, CND-IDS achieves up to 6.1× F-score improvement over the SOTA unsupervised continual learning algorithm.

II. Related Work

Intrusion Detection Systems: Intrusion Detection Systems (IDS) fall into two main categories: Signature-based and Anomaly-based IDS. Signature-based IDS identify attacks by matching network data to known attack patterns. In contrast, Anomaly-based IDS use machine learning methods to analyze statistical properties of network data [16]. Machine learning-based IDS (ML-IDS), a popular form of Anomaly-based IDS, use historical data to train models, enhancing accuracy and reducing the need for expert knowledge [17]. The literature proposes various ML models, including supervised models such as support vector machine, random forest, deep neural networks [18], and unsupervised models like isolation forest, autoencoder, and principal component analysis [14].

Continual Learning for Intrusion Detection: Current ML-IDS solutions are often built on the premise that all data are sampled *i.i.d.* (independent and identically distributed) [5], making them vulnerable to performance degradation when data distributions shift. Continual Learning (CL) seeks to train ML models to acquire new information while minimizing the loss of previously learned knowledge. Prasath et al. [9] highlight that intrusion datasets exhibit these shifting distributions, underscoring the need for CL-capable IDS that can autonomously adapt to the evolving characteristics of the data. Few studies have recently adopted CL for intrusion detection. Kozik et al. [7] extend efficient lifelong learning algorithm [19] to detect cyber attacks and address the data imbalance problem. Kumar et al. [20] applied Elastic Weight Consolidation and Gradient Episodic Memory CL algorithms. In a follow-up study [10], they extend a memory-based CL method and proposed a novel approach (perturbation assistance for parameter approximation), improving CL scalability. Nevertheless, all of these studies leverage labeled attack data to devise their CL solutions, which may not be feasible in IDS due to the difficulty and expense of obtaining such labeled data.

Continual Novelty Detection: Continual Novelty Detection (CND) primarily addresses detecting distribution shifts after ML model deployment. To illustrate, Kim et al. [21] highlight how ND combined with CL is necessary for realistic class incremental learning. Aljundi et al. [22] investigate CND for detecting data outside the current model's knowledge. Finally, Rios et al. [23] propose a novel CND framework for detecting new data and incorporating it into the current ML model's knowledge. These studies focus on using ND to identify data outside the training set and integrate it into the ML model via CL strategies. However, Faber et al. [15] demonstrate the performance decline in ND-algorithms themselves under continual learning settings. Given the constantly changing nature of attacks, it is necessary for ML-IDS systems to continually adapt without labels, which motivates our work. To the best of our knowledge, we are the first to propose continual novelty detection for ML-based intrusion detection.

III. CND-IDS Framework

Fig. 2 illustrates our continual novelty detection-based intrusion detection (CND-IDS) framework. We first apply a continual learning data preparation phase, where we divide the intrusion data into normal data, training experiences, and test experiences. Our framework consists of two main components: a continual feature extractor (CFE) and principal component analysis (PCA)-based novelty detector. The CFE creates a

representation of the data that is used to distinguish normal and attack data by the novelty detector. The CFE allows the model to learn new feature representations overtime, so that it can easily adapt as new attacks are introduced through the data stream. CFE is trained through our continual novelty detection loss function, to continually learn from the input stream without forgetting. The next component, PCA reconstruction, provides an anomaly score on the feature output from the feature extractor. PCA is trained purely on normal data, therefore it can generalize well to unseen attacks. Finally we apply a threshold, τ, such that if the anomaly score is greater than τ, we classify that data as an attack. Overall, our learning framework does not require any labeled attack data.

A. Continual Learning (CL) Data Preparation

We assume a given dataset contains attack data A, normal (benign) data N, and various attack types C. To begin our data preparation, we remove 10% of the normal data, and save this as clean normal data N_c to train PCA novelty detector. For the remaining data, we split it across an arbitrary number of experiences m. We create a set of experiences E, such that each individual experience is $(E_0, E_1, ...E_m)$. Each experience contains a portion of the remaining normal data, of size $\frac{0.9*|N|}{m}$. The attacks are distributed such that each experience contains $\frac{|C|}{m}$ different attack classes. Therefore, each experience contains some number of *unique* attacks to that experience. This allows us to test on zero-day attacks through future experiences, and seen attacks through current or past experiences. Each experience E_i is split into training and testing set. The training split contains only X_{train} data points, meaning it does not include any labels. The test data contains $\{X_{test}, Y_{test}\}$ where $Y_{test} \in \{0, 1\}$, where 0 means normal, and 1 means attack data. For each experience E_i, we utilize the training set X^i_{train} and N_c to train CND-IDS. We then evaluate the model using the test points $\{X_{test}, Y_{test}\}$. This setup is designed to represent a realistic intrusion detection scenario, where attacks are entirely unknown before deployment into a continual learning environment. In our framework, the only known data is N, while all attack data is unlabeled. With this data setup, we can assess model's performance across various attack scenarios. Specifically, we can simulate the model's response to zero-day attacks as well as known attacks.

B. Proposed Algorithm

Algorithm 1 depicts the proposed CND-IDS algorithm, including the training and test steps. The training flow follows three main steps: (i) train the CFE, (ii) encode N_c into features, and (iii) train PCA-based novelty detector on the encoded N_c data. Each of these steps is performed at every training experience. For step (i), the CFE is trained on X_{train} using the loss function from Equation 1. For step (ii), the subset of normal data N_c is encoded by passing it through the trained CFE, where we encode the entire set. Ultimately, in step (iii), we fit the PCA novelty detection model to this encoded normal data. This completes the CND-IDS training, and we then proceed to evaluate on the test experiences.

Algorithm 1 CND-IDS Algorithm (Training and Test)

Require: E - Experience Set; N_c - Subset of Clean Normal Data; CFE - Continual Feature Extractor; ND - PCA Novelty Detector

for $i \leftarrow 1$ **to** $|E|$ **do**
  Get X_{train} from experience data E_i
  Fit CFE to training data X_{train}
  Encode N_c by passing it through CFE
  Fit ND to encoded N_c
  Get $X_{\text{test}}, Y_{\text{test}}$ from all experiences within E
  Encode X_{test} by passing it through CFE
  Use ND to compute anomaly score S_{test} on test data
  Compute threshold τ based on thresholding method
  Compute predictions such that $Y_{\text{pred}} = (S_{\text{test}} > \tau)$
  Compute evaluation metrics based on $Y_{\text{pred}}, Y_{\text{test}}$
end for

At the end of each training experience, we evaluate on the test set of all experiences. Therefore, we evaluate model performance on unseen attacks (future experiences) and seen attacks (past experiences). When evaluating a batch of testing data, we encode all points from X_{test} with CFE. Then use PCA reconstruction to get an anomaly score S_{test} for each point based on the reconstruction loss. Next, we leverage widely used Best-F [24] to select a threshold τ. Ultimately, all anomaly scores greater than τ are classified as attack while all scores less than or equal to τ are classified as normal, resulting in the CND-IDS predictions Y_{pred}. We ultimately compute our evaluation metrics by comparing Y_{pred} with the true labels Y_{test}.

C. Continual Feature Extractor (CFE)

CFE is an autoencoder (AE)-based model that leverages Multi-Layer Perceptron (MLP) as both the encoder and decoder. Our continual novelty detection loss (L_{CND}) consists of a novel cluster separation loss (L_{CS}), a reconstruction loss (L_R), and a continual learning loss (L_{CL}):

$$L_{CND} = L_{CS} + \lambda_R L_R + \lambda_{CL} L_{CL} \tag{1}$$

where $\lambda_R \in [0, 1]$ and $\lambda_{CL} \in [0, 1]$ are hyper-parameters controlling the strength of the reconstruction loss and the continual learning loss respectively.

Cluster Separation Loss: We design a novel cluster separation loss L_{CS} to enhance the performance of our PCA-based novelty detector by increasing the separation between normal and anomalous data points in feature space. To achieve this, we leverage the clean normal data N_c (which is also used to train the PCA model). Using K-Means clustering [25], we identify which points in X_{train} are most similar to N_c and then push these points apart in feature space. Specifically, L_{CS} leverages K-Means clustering to assign pseudo-labels to all points in X_{train}. Then it utilizes triplet margin loss [26] to maximize the Euclidean distance between the different pseudo labels. Calculating the L_{CS} pseudo-labels involves the following steps: 1) Fit K-Means clustering to all points in

X_{train}; 2) Find the cluster labels of all N_c data points; 3) Create the set of "normal data" clusters CL_N, where each cluster contains at least one point from N_c; 4) Assign class 0 to points in X_{train} if their associated cluster is in CL_N, and class 1 otherwise. In summary, after fitting K-Means to X_{train}, we identify the clusters associated with N_c. If a cluster contains any point from N_c, all points in that cluster are assigned to class 0; otherwise, they are assigned to class 1. This effectively splits the data into two, where 0 would represent normal pseudo-label, and 1 would represent anomalous pseudo-label

After assigning the pseudo-labels, the final loss is computed using the widely adopted triplet margin loss [26], defined as:

$$L_{CS} = \max\left(\Delta_{ap} - \Delta_{an} + m, 0\right) \tag{2}$$

where Δ_{ap} represents the Euclidean distance between an anchor point and a positive point (a point of the same class), and Δ_{an} is the Euclidean distance between the anchor and a negative point (a point from a different class). The term $m > 0$ is a hyper-parameter that specifies the desired margin between positive and negative points.

Reconstruction Loss: The reconstruction loss ensures that the features embedded by the encoder retain significant information from the original data. We found that this approach helps the PCA extractor better generalize to future and past experiences by forcing the model to learn encodings that align more closely with the original data. The reconstruction loss is the mean squared error (MSE) between the original input and the reconstructed output of the decoder. Let h be the encoded feature of point x. Then $L_R = MSE(\text{decoder}(h), x)$.

Continual Learning Loss: Our solution to catastrophic forgetting is employing a latent-based regularization loss, L_{CL} [27]. This loss ensures that as the model learns new information, it retains knowledge from past experiences. L_{CL} can be formulated as: $L_{CL} = \sum_{i<c}^{c} MSE(h^c, h^i)$ where c represents the current experience, and h^j is the encoded embedding of input x at experience j. The loss is computed by summing the MSE between the current embedding (h^c) and all previous embeddings h^i where $i < c$. This ensures that the current embedding is still similar to the previous embedding, thereby preventing catastrophic forgetting. To calculate past embedding, we pass the current data point x into a past version of the model. This requires the model to save its state between experiences but does not require it to save any data, which can significantly reduce storage overhead.

D. PCA-based Novelty Detection

Inspired by [23], we create a PCA-based novelty detection approach. Let $\mathbf{h}$ denote the output feature from our feature extractor. We utilize principal component analysis transformation $\mathbf{T}$ to map the input feature from a high dimensional space to low dimensional space $\mathbf{T} : \mathbf{h} \rightarrow \mathbf{l}$. We then utilize the feature reconstruction error (FRE) as the anomaly score: $FRE = \left\|\mathbf{h} - \mathbf{T}^{-1}(\mathbf{l})\right\|^2$, where $\mathbf{T}^{-1}$ is the inverse PCA transform. The anomaly score is therefore the reconstruction loss from this method. We train the PCA transformation on the subset of normal data N_c after it is encoded by the CFE.

IV. Experimental Analysis

A. Experimental Setup

Datasets: Table I provides a detailed breakdown of the SOTA intrusion datasets utilized in our study. For IIoT intrusion, we use IIoT datasets X-IIoTID [28] and WUSTL-IIoT [29]. We also include commonly used network intrusion datasets CICIDS2017 [30] and UNSW-NB15 [31]. For X-IIoTID [28], CICIDS2017 [30], and UNSW-NB15 [31], we split the data across five experiences such that each experience contains two to four attacks. For WUSTL-IIoT [29], we split the data across four experiences such that each experience contains one attack. We perform this data split to simulate an evolving data stream with emerging cyber attacks over time where each experience contains different attacks.

TABLE I: Selected Intrusion Datasets

Dataset	Size	Normal Data	Attack Data	Attack Types
X-IIoTID [28]	820,502	421,417	399,417	18
WUSTL-IIoT [29]	1,194,464	1,107,448	87,016	4
CICIDS2017 [30]	2,830,743	2,273,097	557,646	15
UNSW-NB15 [31]	257,673	164,673	93,000	10

Baselines: We evaluate our algorithm against two SOTA unsupervised continual learning (UCL) algorithms: the Autonomous Deep Clustering Network (**ADCN**) [27], and an autoencoder paired with K-Means clustering. The autoencoder K-Means model is combined with Learning without Forgetting [32] continual learning loss; we refer to this model as **LwF**. Note that both **ADCN** and **LwF** require a small amount of labeled normal and attack data to perform classification. We also compare our approach against SOTA ND methods: local outlier factor (**LOF**) [15], one-class support vector machine (**OC-SVM**) [15], principal component analysis (**PCA**) [23], and Deep Isolation Forest (**DIF**) [33]. Since these ND models cannot be retrained on unlabeled contaminated data, continual learning is not feasible for these methods.

Evaluation Metrics: To evaluate the model performance, we report F_1 score. Since there is a class imbalance within these datasets, to simulate real world IDS, F_1 score gives an accurate idea on attack detection. For the continual learning methods, we evaluate their performance at the end of each training experience on all experience test sets. This generates a matrix of F_1 score results R_{ij} such that i is the current training experience, and j is the testing experience. To summarize this matrix of results, we report widely used CL metrics [34]: average F_1 score on current experience (AVG), forward transfer (FwdTrans), and backward transfer (BwdTrans). For a matrix R_{ij} with m total experiences, our metrics are formulated as follows: $\text{AVG}_{F_1} = \frac{\sum_{i=j} R_{ij}}{m}$; $\text{FwdTrans}_{F_1} = \frac{\sum_{j>i} R_{ij}}{\frac{m*(m-1)}{2}}$; $\text{BwdTrans}_{F_1} = \frac{\sum_{i}^{m} R_{mi} - R_{ii}}{\frac{m*(m-1)}{2}}$. AVG is the average performance on the current test experience at every point of training. FwdTrans is the average performance on "future" experiences, which simulates performance on zero-day attacks. Finally, BwdTrans is the average change in performance of "past" test experiences at a "future" point of training. A negative BwdTrans indicates catastrophic forgetting, whereas a positive

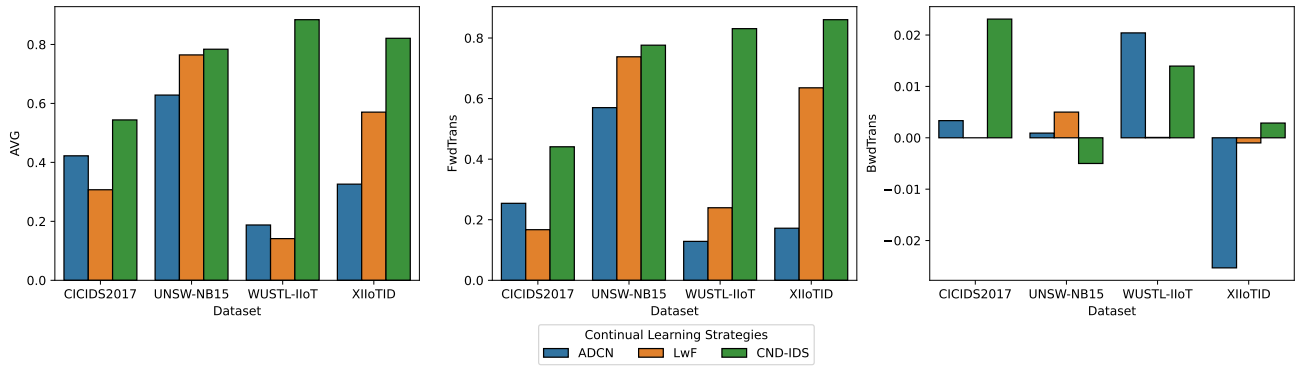


Fig. 3: Continual learning metric results of ADCN [27], LwF [32], and CND-IDS

BwdTrans indicates the model actually improved performance on past experiences after learning a future experience. Overall, AVG measures seen attacks, FwdTrans measures zero-day attacks, and BwdTrans measures forgetting. For all metrics, a higher positive result indicates a better performance.

We also report the threshold-free metric Precision-Recall Area Under the Curve (PR-AUC) [35]. Since CND-IDS requires selecting a threshold, PR-AUC allows us to assess model performance independently of the threshold. We choose PR-AUC over Receiver Operating Characteristic Area Under the Curve (ROC-AUC) because ROC-AUC can give misleadingly high results in the presence of class imbalance [35].

Hyperparameters: We utilize *elbow method* [36] for determining the number of clusters K. We set λ_R and λ_{CL} to 0.1, and for m we use 2 after careful experimentation. For the AE modules of CND-IDS, we use 4-layer MLP with 256 neurons in the hidden layers. We train it using Adam optimizer [37] with a learning rate of 0.001. For PCA, we use the explained variance method and set it to 95% [23].

Hardware: We run our experiments on NVIDIA GeForce RTX 3090 GPU, with a AMD EPYC 7343 16-Core processor.

B. Results

Continual Learning Comparison: Fig. 3 presents the results of our approach CND-IDS compared with ADCN [27] and LwF [32]. CND-IDS shows the best performance on both seen (AVG) and unseen (FwdTrans) attacks across all datasets. CND-IDS also has the highest BwdTrans on all except one dataset (UNSW-NB15). The average BwdTrans of CND-IDS (0.87%) is higher than the average BwdTrans of both ADCN (-0.06%) and LwF (0.09%). Notably, the BwdTrans of CND-IDS is positive for three datasets. Indicating past experiences actually improve after training on future experiences for these datasets. Given the high FwdTrans as well, our approach finds features that generalize well to future experiences.

Table II shows the improvement of CND-IDS over the UCL baselines on all datasets. Bold and underlined cases indicate the best and the second best improvements with respect to each metric, respectively. These improvements were calculated by comparing the performance of CND-IDS to the baselines, where the improvement values represent the proportional increase over the baseline performance. We do not include BwdTrans because a proportional increase does not make sense for a metric that can be negative. CND-IDS has up to $4.50\times$ and $6.1\times$ AVG improvement on ADCN and LwF, respectively. In addition, CND-IDS has up to $6.47\times$ and $3.47\times$ FwdTrans improvement on ADCN and LwF. Averaged across all datasets, CND-IDS shows a $1.88\times$ and $1.78\times$ improvement on AVG, and a $2.63\times$ and $1.60\times$ improvement on FwdTrans, compared to ADCN and LwF, respectively.

TABLE II: CND-IDS Improvement over UCL Baselines

Baseline	Dataset	AVG	FwdTrans
ADCN [27]	X-IIoTID	2.02×	5.00×
	WUSTL-IIoT	**4.50×**	**6.47×**
	CICIDS2017	1.37×	1.73×
	UNSW-NB15	1.29×	1.44×
LwF [32]	X-IIoTID	1.46×	1.35×
	WUSTL-IIoT	**6.11×**	**3.47×**
	CICIDS2017	1.93×	2.64×
	UNSW-NB15	1.11×	1.02×

Overall, these results highlight the benefit of continual ND over UCL methods for IDS. CND-IDS, with its PCA-based novelty detector, excels by effectively harnessing the normal data to identify attacks. A key strength of our approach lies in the assumption that normal data forms a distinct class, while everything else is treated as anomalous. This assumption is particularly well-suited to IDS. In contrast, methods like ADCN and LwF do not make this distinction where they handle both normal and attack data similarly, limiting their ability to fully exploit the inherent structure of the data.

Novelty Detectors Comparison: Fig. 4 compares LOF [15], OC-SVM [15], PCA [23], and DIF [33] with CND-IDS on all datasets. The average F_1 score of the novelty detection methods are compared to the AVG of CND-IDS. It can be seen CND-IDS outperforms all other methods across all datasets. The two best performing methods are DIF and PCA. The average F_1 score improvement across all datasets of CND-IDS is $1.16\times$ and $1.08\times$ over DIF and PCA, respectively. These results highlight the critical role of leveraging information from unsupervised data streams. Unlike these ND algorithms, CND-IDS is capable of continuously learning from this unsupervised data, enabling it to enhance PCA reconstruction over time. By

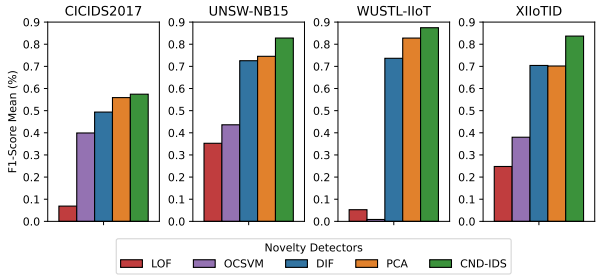


Fig. 4: Average F_1 score on all experiences of CND-IDS and novelty detection methods: LOF, OC-SVM, PCA, DIF

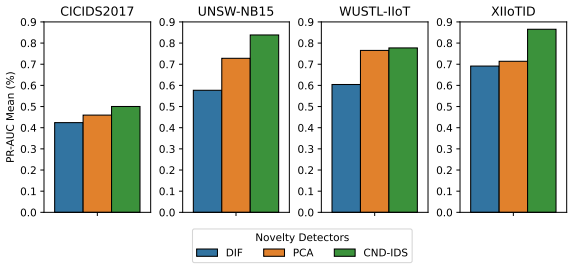


Fig. 5: Thresholding Free Evaluation of CND-IDS

integrating evolving data patterns, CND-IDS not only adapts to new anomalies but also improves its overall detection accuracy, demonstrating a clear advantage in dynamic environments.

Pre-threshold Evaluation: While thresholding plays a crucial role in attack decision-making, evaluating model prediction performance before applying threshold is also important. The UCL algorithms (ADCN [27] and LwF [32]) do not output anomaly scores because they select classes based on the closest labeled cluster. Therefore we compare against the two best ND methods: DIF [33] and PCA [23]. Fig. 5 presents the PR-AUC values of DIF, PCA, and CND-IDS. It can be seen that CND-IDS provides the best threshold free results, which aligns with the threshold-based results presented earlier. The strong performance of CND-IDS in both pre-threshold and threshold-based evaluations demonstrates that the model is robust regardless of the decision threshold.

C. Ablation Study

To demonstrate the impact of our loss function components, we perform an ablation study. Table III shows the results of CND-IDS with each loss function removed to demonstrate their individual effectiveness. Bold and underlined cases indicate the best and the second best performances with respect to each metric, respectively. CND-IDS without reconstruction loss (L_R) and CND-IDS without cluster separation loss (L_{CS}) performs worse in all categories. CND-IDS without both L_R and continual learning loss (L_{CL}) actually performs better AVG but has worse BwdTrans and FwdTrans. AVG does not account for past experiences, so the significantly negative BwdTrans indicates CND-IDS w/o L_R and L_{CL} forgets, and therefore would perform worse on those experiences in the future. This would make sense as a regularization loss to improve continual learning would slightly decrease performance in non-continual scenario. Overall CND-IDS has the best results when taking every metric category into account. Notably the low BwdTrans and FwdTrans of CND-IDS (w/o L_R) showcases how the reconstruction loss helps CND-IDS generalize better to unseen and past data. This highlights the power of L_R to provide good features for continual learning.

TABLE III: Ablation Study of CND-IDS Loss Functions

Strategy	AVG	BwdTrans	FwdTrans
CND-IDS	<u>76.92%</u>	**0.87%**	**73.70%**
CND-IDS (w/o L_{CS})	66.23%	<u>0.09%</u>	70.26%
CND-IDS (w/o L_R)	72.86%	-5.44%	67.82%
CND-IDS (w/o L_R and L_{CL})	**79.92%**	-11.26%	<u>71.01%</u>

TABLE IV: Average inference time (in ms) per test sample

Strategy	CND-IDS	ADCN	LwF	DIF	PCA
Inference Time (ms)	<u>0.0019</u>	0.4061	0.0677	1.0535	**0.0018**

D. Overhead Analysis

Table IV evaluates the inference overhead of CND-IDS compared to ADCN [27], LwF [32], DIF [33], and PCA [23]. CND-IDS offers the fastest inference time among continual learning methods. Out of novelty detection methods, CND-IDS is second only to PCA. We attribute the efficiency of CND-IDS to avoiding the clustering classification used by LwF and ADCN. The difference between CND-IDS and PCA is minimal, only 0.0001 milliseconds slower, due to the additional but lightweight step of encoding the data. Considering that the average median flow duration across datasets is 27.77 milliseconds, the overhead introduced by CND-IDS is negligible in the context of real-time traffic flow.

V. Conclusion

IDS are essential for ensuring a robust and comprehensive security strategy. ML methods have been recently adopted for this task due to their great prediction performance. However, current ML-IDS solutions are vulnerable to under-represented attacks and constantly changing environments. Therefore, future IDS must be able to (i) generalize to unseen attacks and (ii) adapt to evolving attacks over time. To address these challenges, we present CND-IDS, a continual novelty detection based IDS algorithm. CND-IDS combines PCA novelty detection with a continual feature extractor. CND-IDS is capable of continually learning with no attack labels, allowing it to detect attacks in a changing data stream. We have demonstrated that on realistic intrusion datasets, CND-IDS is up to 6.1× more effective than SOTA unsupervised CL method for detecting seen attacks and up to 6.5× more effective at detecting future attacks.

Acknowledgments

This work has been funded in part by NSF, with award numbers #1826967, #1911095, #2003279, #2052809, #2100237, #2112167, #2112665, and in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA.

REFERENCES

[1] I. Kumar, "Emerging threats in cybersecurity: A review article," *International Journal of Applied and Natural Sciences*, vol. 1, p. 01–08, Jul. 2023.

[2] H.-J. Liao *et al.*, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[3] O. Gungor, T. Rosing, and B. Aksanli, "Roldef: Robust layered defense for intrusion detection against adversarial attacks," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2024.

[4] "Study: Hackers attack every 39 seconds." https://eng.umd.edu/news/story/study-hackers-attack-every-39-seconds. Accessed: 2023-01-11.

[5] E. Verwimp *et al.*, "Continual learning: Applications and the road forward," *arXiv preprint arXiv:2311.11908*, 2023.

[6] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: theory, method and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[7] R. Kozik *et al.*, "Balanced efficient lifelong learning (b-ella) for cyber attack detection.," *J. Univers. Comput. Sci.*, vol. 25, no. 1, pp. 2–15, 2019.

[8] R. R. Karn, P. Kudva, and I. M. Elfadel, "Learning without forgetting: A new framework for network cyber security threat detection," *IEEE Access*, vol. 9, pp. 137042–137062, 2021.

[9] S. Prasath *et al.*, "Analysis of continual learning models for intrusion detection system," *IEEE Access*, vol. 10, pp. 121444–121464, 2022.

[10] S. kumar Amalapuram, S. S. Channappayya, and B. Tamma, "Augmented memory replay-based continual learning approaches for network intrusion detection," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[11] C. Oikonomou *et al.*, "A multi-class intrusion detection system based on continual learning," in *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, pp. 86–91, IEEE, 2023.

[12] S. Channappayya, B. R. Tamma, *et al.*, "Augmented memory replay-based continual learning approaches for network intrusion detection," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[13] M. Macas, C. Wu, and W. Fuertes, "A survey on deep learning for cybersecurity: Progress, challenges, and opportunities," *Computer Networks*, vol. 212, p. 109032, 2022.

[14] M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Towards model generalization for intrusion detection: Unsupervised machine learning techniques," *Journal of Network and Systems Management*, vol. 30, pp. 1–25, 2022.

[15] K. Faber, R. Corizzo, B. Sniezynski, and N. Japkowicz, "Lifelong continual learning for anomaly detection: New challenges, perspectives, and insights," *IEEE Access*, vol. 12, p. 41364–41380, 2024.

[16] N. Gupta, V. Jindal, and P. Bedi, "A survey on intrusion detection and prevention systems," *SN Comput. Sci.*, June 2023.

[17] O. Gungor, E. Li, Z. Shang, Y. Guo, J. Chen, J. Davis, and T. Rosing, "Rigorous evaluation of machine learning-based intrusion detection against adversarial attacks," in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, pp. 152–158, IEEE, 2024.

[18] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *applied sciences*, vol. 9, no. 20, p. 4396, 2019.

[19] P. Ruvolo and E. Eaton, "Ella: An efficient lifelong learning algorithm," in *International conference on machine learning*, pp. 507–515, PMLR, 2013.

[20] S. K. Amalapuram *et al.*, "Continual learning for anomaly based network intrusion detection," in *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pp. 497–505, IEEE, 2022.

[21] G. Kim, C. Xiao, T. Konishi, Z. Ke, and B. Liu, "Open-world continual learning: Unifying novelty detection and continual learning," 2023.

[22] R. Aljundi, D. O. Reino, N. Chumerin, and R. E. Turner, "Continual novelty detection," in *Conference on Lifelong Learning Agents*, pp. 1004–1025, PMLR, 2022.

[23] A. Rios, N. Ahuja, I. Ndiour, U. Genc, L. Itti, and O. Tickoo, "incdfm: Incremental deep feature modeling for continual novelty detection," in *European Conference on Computer Vision*, pp. 588–604, Springer, 2022.

[24] Y. Su *et al.*, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2828–2837, 2019.

[25] D. Seo, S. Kim, S. Oh, and S.-H. Kim, "K-means clustering-based safety system in large-scale industrial site using industrial wireless sensor networks," *Sensors*, vol. 22, no. 8, 2022.

[26] F. Schroff *et al.*, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

[27] A. Ashfahani and M. Pratama, "Unsupervised continual learning in streaming environments," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 12, pp. 9992–10003, 2023.

[28] M. Al-Hawawreh *et al.*, "X-iiotid: A connectivity-agnostic and device-agnostic intrusion data set for industrial internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3962–3977, 2021.

[29] M. Zolanvari *et al.*, "Wustl-iiot-2021 dataset for iiot cybersecurity research." Washington University in St. Louis, USA, oct 2021. http://www.cse.wustl.edu/~jain/iiot2/index.html.

[30] I. Sharafaldin *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy*, 2018.

[31] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, IEEE, 2015.

[32] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, 2018.

[33] H. Xu, G. Pang, Y. Wang, and Y. Wang, "Deep isolation forest for anomaly detection," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–14, 2023.

[34] N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni, "Don't forget, there is more than forgetting: new metrics for continual learning," *arXiv preprint arXiv:1810.13166*, 2018.

[35] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, (New York, NY, USA), p. 233–240, Association for Computing Machinery, 2006.

[36] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

POLARIS: Explainable Artificial Intelligence for Mitigating Power Side-Channel Leakage

Tanzim Mahfuz[1], Sudipta Paria[2], Tasneem Suha[1], Swarup Bhunia[2], and Prabuddha Chakraborty[1]

[1]Department of Electrical & Computer Engineering, University of Maine, Orono, ME, USA
[2]Department of Electrical & Computer Engineering, University of Florida, Gainesville, FL, USA

***Abstract*—Microelectronic systems are widely used in many sensitive applications (e.g., manufacturing, energy, defense). These systems increasingly handle sensitive data (e.g., encryption key) and are vulnerable to diverse threats, such as, power side-channel attacks, which infer sensitive data through dynamic power profile. In this paper, we present a novel framework, `POLARIS` for mitigating power side channel leakage using an Explainable Artificial Intelligence (XAI) guided masking approach. `POLARIS` uses an unsupervised process to automatically build a tailored training dataset and utilize it to train a masking model. The `POLARIS` framework outperforms state-of-the-art mitigation solutions (e.g., VALIANT) in terms of leakage reduction, execution time, and overhead across large designs.**

***Index Terms*—Power side-channel Analysis and Mitigation, Hardware Security, Explainable Artificial Intelligence.**

I. Introduction

Edge computing devices in Internet-of-Things (IoT) applications are increasingly used in diverse applications from domains, such as, Industry 4.0, surveillance, smart cities, defense, healthcare, and aerospace. These devices often record and process sensitive data and hence become a target for diverse attacks from malicious entities. A malicious entity can attempt to leak sensitive information from these devices through various means, such as timing side-channel attacks, power side-channel attacks, and fault-injection attacks.

In this work, we focus on mitigating power side-channel attacks [1], [2], which involve measuring and analyzing the power consumption traces of a system to infer the values of sensitive data. This inference is possible because different computing instructions consume different amount of power while running on an unprotected/unmasked electronic hardware. Wide array of techniques have been developed to mitigate this concern, which include:

1) **Quantifying Power Side Channel Leakage**: Techniques such as Test Vector Leakage Assessment (TVLA) were developed to estimate power side channel leakage at a gate level for a certain number of traces [3].
2) **Masking Gates**: Composite logic gates were developed to replace traditional gates to provide better power side channel leakage camouflaging [4], [5].
3) **Efficient Utilization of Masking Gates**: Brute force replacement of all gates using composite masking gates can lead to very high design overheads (area, power, delay). Hence, techniques such as VALIANT [6] were developed to perform selective masking to obtain a good masking effect at a lower overhead.

Existing power side-channel countermeasures, such as, [6], [7], [8], which relies on fixed heuristics (static algorithm), may not perform well for diverse designs. Additionally, relying on TVLA (a simulation-based approach) analysis makes it highly time-consuming to operate on large designs. DL [9] and LLM-based [10] leakage estimation techniques are proposed but are inadequate in leakage mitigation due to their high training times and lack of synthetic data support or explainability features. To mitigate these shortcomings, we present a novel Explainable AI based Design-for-Security framework `POLARIS` (POwer Side-Channel LeAkage Reduction using Explainable AI Solution), that utilizes a search process to automatically learn how to most optimally insert masking gates to achieve high power side channel leakage reduction at low overhead. We implement `POLARIS` with training data generated using smaller open-source designs and extensively evaluated its effectiveness across various designs with varying complexities. `POLARIS` is able to outperform VALIANT [6] in terms of performance (execution time), effectiveness (overall leakage reduction), and design overhead (area, power, delay). `POLARIS` does not rely on TVLA for leakage estimation and mitigation through masking and is approximately 6× faster than VALIANT, making it scalable for larger designs. In summary, we make the following key contributions:

1) Developed a novel explainable design-for-security (DFS) framework (`POLARIS`) for mitigating power side-channel vulnerabilities in electronic systems.
2) Designed an unsupervised algorithm to automatically generate training data for `POLARIS` using a search approach and an AI algorithm for efficiently masking a given digital design.
3) Implemented the `POLARIS` framework as a parameterized tool & integrated it into the ASIC design flow.
4) Extensively evaluated `POLARIS` (different settings) and compared it with VALIANT [6], a state-of-the-art alternative solution, using more than 10 large digital designs.

The rest of the paper is organized as follows: Section II describes the brief background and related work, followed by the motivation behind this work in Section III. Section IV outlines the methodology, and Section V demonstrates the experimental results. We conclude the paper in Section VI.

II. Background

A. Test Vector Leakage Assessment (TVLA)

TVLA [3] is a widely recognized method for power side-channel analysis based on Welch's t-test. The approach captures the DUT's leakage behavior under varied inputs to detect any differences. Mathematically, let us denote two sets of data by $\mathcal{Q}_0$ and $\mathcal{Q}_1$, their cardinality by n_0 and n_1, respectively. Let also μ_0 (resp. μ_1) and s_0^2 (resp. s_1^2) denote sample mean and sample variance of the set $\mathcal{Q}_0$ (resp. $\mathcal{Q}_1$). The t-test statistic and the degree of freedom v are computed as:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{(\frac{s_0^2}{n_0}) + (\frac{s_1^2}{n_1})}} \qquad v = \frac{(\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1})^2}{\frac{(\frac{s_0^2}{n_0})^2}{n_0 - 1} + \frac{(\frac{s_1^2}{n_1})^2}{n_1 - 1}} \tag{1}$$

The t-statistic is typically assessed with a threshold of $\pm$ 4.5 for distinguishability. This threshold ensures that for $|t| > 4.5$ and $v > 1000$, the p-value drops below 0.00001, indicating over 99.999% confidence against the null hypothesis [11]. A crypto implementation is considered protected with 99.999% confidence if $|t| \leq 4.5$; otherwise, it may indicate leakage [6]. The test can be performed in two ways: (i) Fixed-vs-Fixed (using known intermediate values) or (ii) Fixed-vs-Random (using fixed and random input patterns) [12].

The naive algorithm to compute variance (s_0^2, s_1^2) followed by the respective standard deviation (ρ_0, ρ_1) is inefficient as it involves processing full trace points twice due to the dependency on mean (μ_0, μ_1) values for the computation. The evaluation for ρ_0 is given as:

$$\rho_0 = \frac{\sum_i (x_i - \mu_0)^2}{n_0}, \text{ where } x_i \in \mathcal{Q}_0 \tag{2}$$

TVLA trace collection is slow due to repeated mean and variance calculations. To accelerate it, [11] proposed an efficient one-pass method for raw and central moments computation during trace acquisition. The raw moment (sample mean) and central moment (variance) of order $d = 1$ for an expanded set $\mathcal{Q}' = \mathcal{Q} \cup y$ with n elements are calculated as:

$$M_1^{\mathcal{Q}'} = M_1^{\mathcal{Q}} + \frac{\Delta}{n}, \text{ where } \Delta = y - M_1^{\mathcal{Q}} \tag{3}$$

$$\mu = M_1 \text{ and } s^2 = CM_2, \text{ where } CM_2 = M_2 - M_1^2 \tag{4}$$

This method can be extended to compute the central moments at any $d > 1$ using the equations as given in [11].

B. Masking

Masking is a well-known countermeasure for mitigating side-channel attacks [13]. Masking randomizes sensitive data in cryptographic operations using a secret sharing scheme at the logic level. To achieve d-th order security, each variable x is split randomly into $(d + 1)$ shares, such that $x = x^{(1)} \oplus x^{(2)} \oplus \cdots \oplus x^{(d+1)}$, where $x^{(i)}$ denotes i^{th} share of x and $\oplus$ denotes XOR operation. This provides significant protection since that adversary would need to recover all $d+1$ shares to obtain the actual value of x, which requires statistical analysis up to d^{th} order. Authors in [4], [5] proposed efficient masking schemes for block cipher implementations against DPA attacks. As example, the *Masked AND* and *OR* operation proposed by [4] is demonstrated in Fig. 1 and described below:

If x_i and y_j denote the bits that mask the 'real' bits a_i and b_j then we denote masked bits as $\hat{a} = (a_i \oplus x_i)$ and $\hat{b} = (b_j \oplus y_j)$ correspondingly. We introduce a random bit, z, as a new mask for computing a *masked AND* operation as:

$$\mathcal{M}(a \cdot b) = ((\hat{a} \cdot \hat{b}) \oplus ((x_i \cdot \hat{b}) \oplus ((x_i \cdot y_j) \oplus z))) \oplus (y_j \cdot \hat{a}) \tag{5}$$

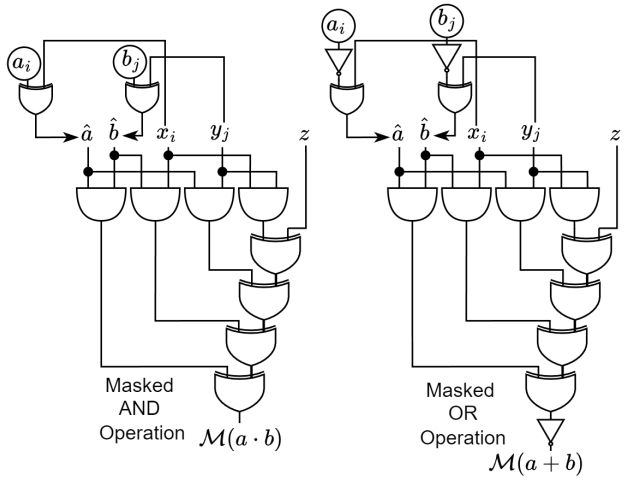


Fig. 1: *AND* and *OR* operation on masked data.

C. Related Work

Existing power side-channel leakage evaluation/mitigation approaches like CASCADE [7], Karna [8], VALIANT [6] incorporate TVLA analysis and hence suffers from high runtime and scalability issues. AI-based methods like DL-LA [9] and Netlist Whisperer [10] are often limited by the lack of training data and are vulnerable to adversarial attacks due to reliance on external training data [14], [15], [16], [17].

`POLARIS` stands out by using a synthetic data generation approach for creating its own training data, ensuring improved performance and resilience against adversarial attacks. `POLARIS` also incorporates XAI-based rules generation for efficient power side channel mitigation through masking. Table I provides a comparative analysis between the existing solutions and the proposed framework.

III. Motivation

A. Can Explainable AI Generate Custom DFS Rules/Models for Power Side Channel Mitigation?

Power side channel mitigation techniques such as Karna and VALIANT rely on static algorithms, in other words, a fixed set of hand-crafted rules. This is a limiting factor and prevents optimal operation on a diverse set of designs. We hypothesize that an Explainable AI (XAI) framework might be able to generate custom rules/models (to mitigate power side channel leakage) based on a given type of design, making the process more autonomous and adaptive.

B. Can We Bypass TVLA using AI?

Most of the prevalent power side channel mitigation frameworks (e.g., Karna, VALIANT) rely on TVLA for estimating

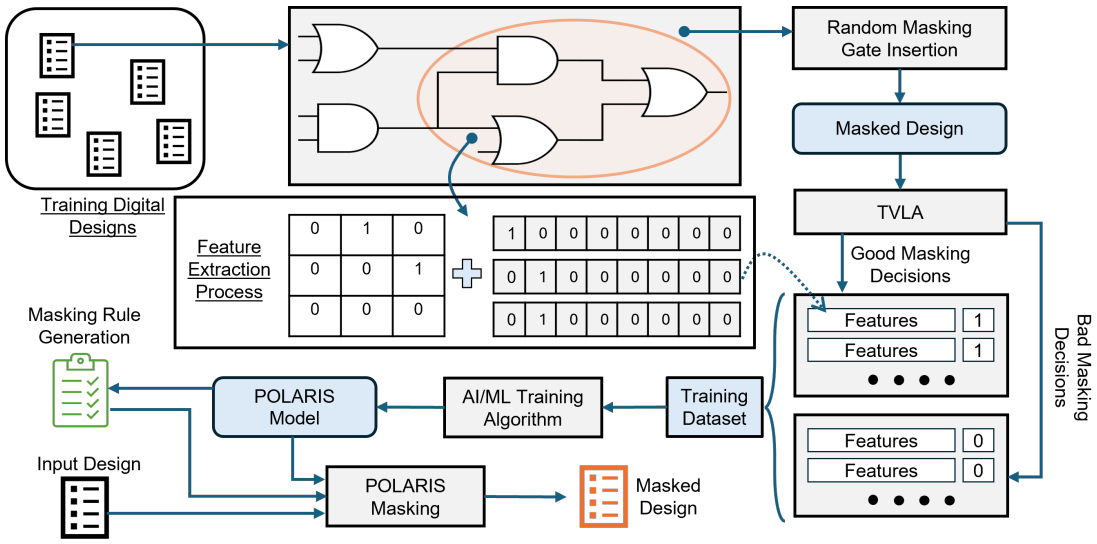


Fig. 2: Overview of the POLARIS framework.

power leakage. TVLA has major scalability concerns particularly for large industry-grade designs that in turn hampers the scalability of frameworks such as Karna and VALIANT. We hypothesize, AI techniques might be able to predict the leakage (directly or in an integrated way) associated with individual gates faster and with a linear time complexity (with respect to design size).

C. Can We Generate the Training Data?

Training data is a major limiting factor for AI frameworks in the hardware security domain. Techniques such as DL-LA [9] and Netlist Whisperer [10] require a very high amount of data making it almost infeasible to effectively scale across different design types. If we can develop a framework that can leverage synthetic training data automatically generated via an unsupervised technique, then it would be highly scalable across diverse design types.

IV. Methodology

We build on the motivations presented in Sec. III to design a framework that can: (1) Leverage XAI for creating custom power side channel mitigation rules/models; (2) Bypass TVLA using an AI approach; (3) Utilize automatically generated synthetic data for model training. The overall framework, shown in Fig. 2 operates in three major stages: (i) Knowledge extraction, data generation, and building the ML model; (ii) Interpreting the model using the explainable SHAP [18] framework; and (iii) Masking against side-channel vulnerabilities.

A. Knowledge Extraction and ML Model Development

Obtaining a large dataset for training AI models is a major concern particularly in the domain of hardware security and EDA. To address this concern, POLARIS utilizes a novel unsupervised approach to generate a large database for AI model training. As shown in Fig. 2, the framework POLARIS converts any digital design represented as gate-level netlist ($\mathcal{D}$) into a graph (G_r) such that $G_r = (V, E)$ where, V: gates and E: interconnections and randomly inserts masking gates based on mask size ($\mathcal{M}_{size}$). Then, POLARIS calculates leakage values ($\mathbb{L}_G$) using TVLA analysis and compares it to the original leakage values. If the difference exceeds a threshold (θ_r), the masking is labeled 'good' (1); otherwise, 'bad' (0) and appended to $\{X_data, Y_data\}$. These labels are linked to structural features used for knowledge extraction. The framework employs local structural features for training and evaluation, similar to [19],[20]. The structural features of a gate include information such as their local placement and interconnections. In a sub-design graph, gate connectivity

TABLE I: Comparing POLARIS with existing solutions for power side-channel leakage assessment and mitigation.

Approach	Method	Model Training		Feature Set	Mitigation Support	Performance	Platform
CASCADE [7]	TVLA	N/A		N/A	No	Slow	ASIC
Karna [8]	TVLA	N/A		N/A	Limited	Slow	ASIC
VALIANT [6]	TVLA	N/A		N/A	Yes	Slow	ASIC
DL-LA [9]	DL	Training Time: High Adversarial Attacks: Possible	Explainability: No Synthetic Data Support: No	Trace based	No	Slow	ASIC/ FPGA
Netlist Whisperer [10]	LLM	Training Time: High Adversarial Attacks: Possible	Explainability: No Synthetic Data Support: No	ANF equations	Yes	Slow	ASIC
POLARIS (This work)	XAI	Training Time: Low Adversarial Attacks: No	Explainability: Yes Synthetic Data Support: Yes	Structural Analysis	Yes	Fast	ASIC/ FPGA*

*POLARIS can be extended to FPGA design flow by re-training the model with lookup table (LUT) based FPGA netlists.

Algorithm 1: Cognition Generation

Input: $\mathcal{D}, \mathcal{M}_{size}, \mathcal{L}, itr, \theta_r$
Output: $\mathbb{M}$

$G_r \leftarrow$ ***graphify***$(\mathcal{D})$
$\mathbb{L}_G \leftarrow$ ***leak_estimate***$(\mathcal{D})$
$X_data, Y_data \leftarrow \emptyset$; $run \leftarrow 0$
$\mathcal{R}_{gates} \leftarrow G_r.gates$
while $\mathcal{M}_{size} \leq len(\mathcal{R}_{gates})$ *and* $run \leq itr$ **do**
  $\mathcal{S}_{gates} \leftarrow$ ***random***$(\mathcal{M}_{size}, \mathcal{R})$
  $\mathcal{D}_{mod} \leftarrow$ ***modify***$(\mathcal{S}_{gates}, \mathcal{D})$
  $\mathcal{R}_{gates} \leftarrow \mathcal{R}_{gates} - \mathcal{S}_{gates}$
  $\mathbb{L}_G^{mod} \leftarrow$ ***leak_estimate***$(\mathcal{D}_{mod})$
  for i ***in*** $\mathcal{S}_{gates}$ **do**
    $S_f \leftarrow$ ***structural_features***$(G_r, \mathcal{L}, i)$
    $r_{Ratio} \leftarrow$ ***compare***$(\mathbb{L}_G[i], \mathbb{L}_G^{mod}[i])$
    if $r_{Ratio} \geq \theta_r$ **then**
      $label \leftarrow 1$
    else
      $label \leftarrow 0$
    $X_data.append(S_f)$
    $Y_data.append(label)$
  $run \leftarrow run + 1$
$\mathbb{M} \leftarrow$ ***train_data***(X_data, Y_data)
return $\mathbb{M}$

is encoded with an adjacency matrix and one-hot encoding. Fig. 2 illustrates the structural feature extraction process, showing how gates are vectorized and how the dataset is created via feature extraction and labeling. Breadth-first search (BFS) is employed to explore neighboring gates (Locality $\mathcal{L}$) and assign labels based on leakage comparison. The random insertion process runs iteratively ($\leq itr$ times) until the termination condition is met. Finally, we obtain a trained model ($\mathbb{M}$) trained using $\{X_data, Y_data\}$ that will be used to generate human-understandable rules through model interpretability in the next stage. Algorithm 1 describes the sequence of steps involved in extracting knowledge through structural features and developing the ML model.

B. Explainable AI (XAI): Model Interpretability by SHAP

XAI enhances transparency and accountability in AI by providing methods for designing inherently interpretable models and for analyzing decisions after training. We incorporate the SHAP (SHapley Additive exPlanations) [18] algorithm into our framework to analyze the trained models and extract the underlying model rules driving the mitigation process. SHAP leverages game theory to quantify the contribution of each feature in model's prediction. For an individual prediction, the Shapley value of a feature f, is denoted as ϕ_f, and is computed using Equation 6. Given h total features, there are $h!$ possible permutations (coalitions) of these features. Let g represent a specific coalition that excludes feature f. The weighting factor $\frac{|g|!(h-|g|-1)!}{h!}$ accounts for the number of ways to form such coalitions, and the term $\text{val}(g \cup \{f\}) - \text{val}(g)$ signifies the

Algorithm 2: POLARIS Masking

Input: $\mathcal{D}, \mathbb{M}, \mathcal{R}_L, \mathcal{M}_{size}, \mathcal{L}$
Output: $\mathcal{D}_{\mathcal{M}}$

$G_r \leftarrow$ ***graphify***$(\mathcal{D})$
$\mathcal{G} \leftarrow G_r.gates$
$\mathcal{C} \leftarrow \emptyset$
for i ***in*** $\mathcal{G}$ **do**
  $S_f \leftarrow$ ***structural_features***$(G_r, \mathcal{L}, i)$
  $\mathcal{P}_{pred} \leftarrow$ ***inferencing***$(\mathbb{M}, R_L, S_f)$
  $\mathcal{C}.append(\mathcal{P}_{pred}, i)$
$\mathcal{C}_{top} \leftarrow$ ***sort_descending***$(\mathcal{C})$
$\mathcal{D}_{\mathcal{M}} \leftarrow$ ***modify***$(\mathcal{D}, \mathcal{C}_{top}, \mathcal{M}_{size})$
$\mathbb{L}_{info} \leftarrow$ ***leak_estimate***$(\mathcal{D}_{\mathcal{M}})$
return $\mathcal{D}_{\mathcal{M}}$

marginal contribution of feature f to coalition g. N denotes set of all features $\{1, 2, ...h\}$.

While there are other XAI frameworks, such as LIME [21] and Captum [22], we focus on SHAP due to its versatility. SHAP offers both model-agnostic methods (e.g., Kernel SHAP) and model-specific methods (e.g., Tree SHAP), making it suitable for a broad range of models. The automated rules, unlike handcrafted ones, can be used independently to make masking decisions or alongside the model to achieve better predictions and decisions, as shown in Fig. 2.

$$\phi_f = \sum_{g \subseteq N \setminus \{f\}} \frac{|g|!\,(h - |g| - 1)!}{h!} \left[\text{val}\left(g \cup \{f\}\right) - \text{val}(g)\right] \tag{6}$$

C. POLARIS Masking in Side-Channel Defense

The trained model can be used directly to mask a design for side-channel defense and can also leverage human-readable rules ($\mathcal{R}_L$) rules generated through the XAI framework to improve decision-making. For each gate in the graph (G_r), generated from ($\mathcal{D}$), structural features (S_f) are extracted, and predictions ($\mathcal{P}_{pred}$) are made using the model or rules followed by appending $\mathcal{P}_{pred}$ values to choices ($\mathcal{C}$). The set $\mathcal{C}$ is sorted in descending order to identify the top selections ($\mathcal{C}_{top}$). Finally, masking is applied to the design based on $\mathcal{C}_{top}$, and leakage of the masked design ($\mathcal{D}_{\mathcal{M}}$) is measured. Algorithm 2 outlines all the steps involved.

V. Results and XAI Analysis

In this section, we analyze the effectiveness of the proposed framework POLARIS in identifying and mitigating leakage from open-source designs with varying complexities and leveraging the SHAP framework to mitigate leakage through masking against power side-channel attacks.

A. Experiment Configuration

We utilize knowledge obtained from training on smaller designs to generalize to larger and entirely unseen designs, leveraging the transfer learning approach. The model is trained on smaller open-source designs that can adapt to larger designs, capturing patterns that can be transferred for practical application of transfer learning. We used six designs from the

TABLE II: Quantitative comparison between VALIANT [6] & POLARIS in terms of leakage reduction & runtime efficiency.

Benchmarks	Leakage Value (Per Gate)					Total Leakage Reduction (%)				Time (s)	
	Before	VALIANT	POLARIS			VALIANT	POLARIS			VALIANT	POLARIS
			50%Mask	75%Mask	Full Mask		50%Mask	75%Mask	Full Mask		
des3	2.66	1.20	1.23	1.09	1.02	54.89	53.76	59.02	61.65	38.68	16.40
arbiter	1.42	1.10	1.15	0.98	0.88	22.54	19.01	30.99	38.03	97.62	15.11
sin	2.16	1.23	1.19	1.04	0.91	43.06	44.91	51.85	57.87	42.42	15.21
md5	2.93	1.39	1.41	1.28	1.21	52.56	51.88	56.31	58.70	202.11	59.11
voter	2.96	1.30	1.12	1.10	1.01	56.08	62.16	62.84	65.88	84.59	28.36
square	2.94	1.23	1.29	1.25	1.21	58.16	56.12	57.48	58.84	309.00	56.62
sqrt	3.04	1.44	1.48	1.31	1.20	52.63	51.32	56.91	60.53	281.05	75.53
div	2.28	1.44	1.37	1.28	1.18	36.84	39.91	43.86	48.25	389.26	92.51
memctrl	1.62	1.21	1.18	1.03	1.00	25.31	27.16	36.42	38.27	2257.50	94.44
multiplier	2.44	1.23	1.22	1.14	1.12	49.59	50.00	53.28	54.10	498.15	127.03
log2	2.25	1.16	1.20	1.11	1.06	48.44	46.67	50.67	52.89	428.31	148.38
Average	**2.43**	**1.27**	**1.26**	**1.15**	**1.07**	**45.46**	**45.72**	**50.88**	**54.09**	**420.79**	**66.25**

Note: **'X% Mask'** denotes X% of total number of leaky gates. The computational time for POLARIS with varying mask sizes is similar.

TABLE III: Comparison among different ML models used in POLARIS. Values indicate leakage reduction in %.

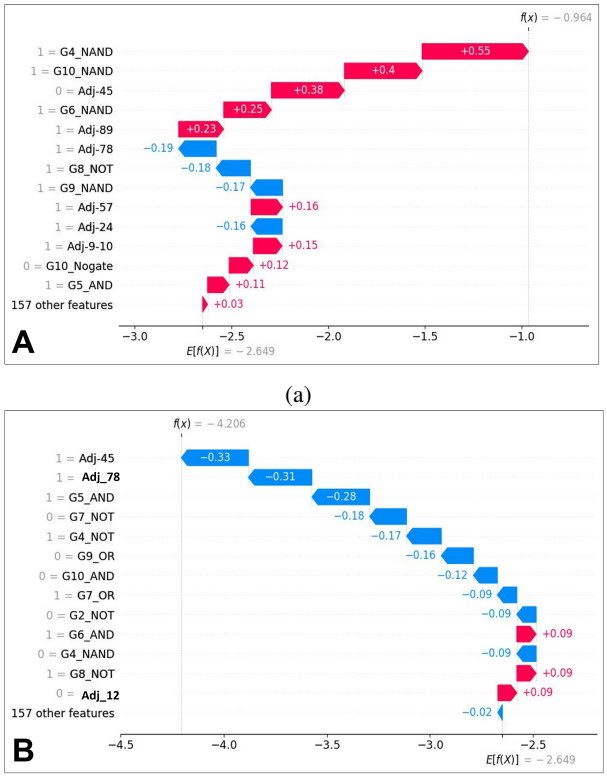
Designs	Random Forest	XGBoost	AdaBoost
des3	33.73	41.35	61.65
arbiter	29.36	43.02	38.03
sin	32.48	56.89	57.87
md5	43.97	54.70	58.70
voter	58.67	62.39	65.88
square	22.37	42.96	58.84
sqrt	39.02	53.57	60.53
div	68.69	74.36	48.25
memctrl	18.95	28.16	38.27
multiplier	57.09	55.09	54.10
log2	57.37	53.88	52.89
Average	**41.97**	**51.49**	**54.09**

Note: We observe nominal differences in computational time across models.

ISCAS-85 benchmark suite [23] synthesized using Synopsys Design Compiler (DC) for training and simulated using 10,000 traces for leakage measurement by TVLA analysis. Selecting smaller designs helps reduce the model's training time (∼40 minutes). The key parameters are: $\mathcal{M}_{size} = 200$, $\mathcal{L} = 7$ (considering 7 neighboring gates), $itr = 100$, and $\theta_r = 0.70$ (indicating a leakage reduction of 70% or more as good masking). The rationale behind selecting the value of θ_r to 0.70 is that selecting higher values lead to significant data imbalance, which could cause the model to underfit and hinder its ability to generalize effectively. The evaluation phase includes larger and entirely different designs with varying $\mathcal{M}_{size}$ from EPFL [24] and MIT-CEP [25] benchmark suite that are distinct from training designs.

B. Comparing ML Models for Leakage Reduction

In this work, we explore several machine-learning techniques to develop improved masked designs that mitigate side-channel vulnerabilities. Table III compares the leakage reduction by Random Forest, XGBoost, and AdaBoost models for $\mathcal{L} = 7$, $\theta_r = 0.7$, and $\mathcal{M}_{size}$ denoting number of leaky gates identified by TVLA. We applied SMOTE [26] for Random Forest and employed weighted training for XGBoost and AdaBoost models for handling imbalance that occurred due to θ_r. The learning rate (α) for both XGBoost and AdaBoost was set to 0.01. AdaBoost outperforms the other models with a 54.09% leakage reduction on average, making it the chosen model for the remaining experiments.



(a)

(b)

Fig. 3: SHAP waterfall plots generated by POLARIS Adaboost (ADB) model for power side channel defense.

C. Comparison with State-of-the-Art

Table II compares the performance of VALIANT framework [6] and POLARIS in terms of leakage per gate, total leakage reduction, and execution time. POLARIS is tested with 50%, 75%, and full (100%) masking sizes. Even with only 50% masking, POLARIS achieves a 45.72% leakage reduction, surpassing VALIANT's 45.46% with full masking. At 75% and full masking, POLARIS reduces leakage by 50.88% and 54.09%, respectively, outperforming VALIANT. Additionally,

TABLE IV: Comparison of area, power, and delay overheads between VALIANT [6] and POLARIS[§].

Designs	Original			VALIANT (x Original[◇])			POLARIS[§] (x Original[◇])			Reduction (%) in POLARIS		
	Area (μM²)	Power (mW)	Delay (ns)	Area	Power	Delay	Area	Power	Delay	Area	Power	Delay
des3	9083.31	2.73	1.3	3.7	3.1	1.6	2.4	1.8	1.2	35.14	41.94	25.00
arbiter	10310.05	1.61	1.23	3.1	3.3	3.3	2.1	2	2.1	32.26	39.39	36.36
sin	13421.04	9.12	7.98	3.6	3.1	3.1	2.3	1.6	1.9	36.11	48.39	38.71
md5	24217.29	0.94	0.12	5.1	3.8	1.4	3.1	1.7	1.1	39.22	55.26	21.43
voter	28090.42	17.25	2.53	4.2	3.8	4.5	2.5	2.4	2.9	40.48	36.84	35.56
square	51117.09	32.98	9.84	4.7	3.8	3.2	2.7	2.3	1.2	42.55	39.47	62.50
sqrt	42921.71	83.28	279.99	5.3	3.8	3	3.2	2.3	1.9	39.62	39.47	36.67
div	44048.97	12.59	196.90	3.1	2.9	2.2	2.5	2.2	1.7	19.35	24.14	22.73
memctrl	42921.71	8.21	3.55	4	4.2	2.6	2.2	2.1	1.9	45.00	50.00	26.92
multiplier	66185.85	56.62	10.13	2.4	1.9	2	2.1	1.5	1.7	12.50	21.05	21.68
log2	77395.08	66.95	16.55	3.9	3.2	3.4	2.4	1.6	2.1	38.46	50.00	38.24
Average	**37246.59**	**26.57**	**48.19**	**3.92**	**3.35**	**2.75**	**2.50**	**1.95**	**1.79**	**34.61**	**40.54**	**33.25**

§ We utilize POLARIS w/ 50% Mask for comparison with VALIANT, achieving a comparable leakage reduction while masking half the number of gates.
◇ Overheads are reported as x times original value.

TABLE V: Power side-channel mitigation rules generated via the POLARIS framework (AdaBoost Model).

Rules	As long as	Procedure
A	G4 = NAND && G5 = AND && G4 (NAND) and G5 (AND) are not connected && G6 = NAND && G10 = NAND && G8 (NOT) and G9 (NAND) are connected	Select & Replace with masking gate
B	G5 = AND && G4 = NOT && G7 = OR && G4 (NOT) and G5 (AND) are connected blue && G7 (OR) and G8 (NOT) are connected	Do not Mask

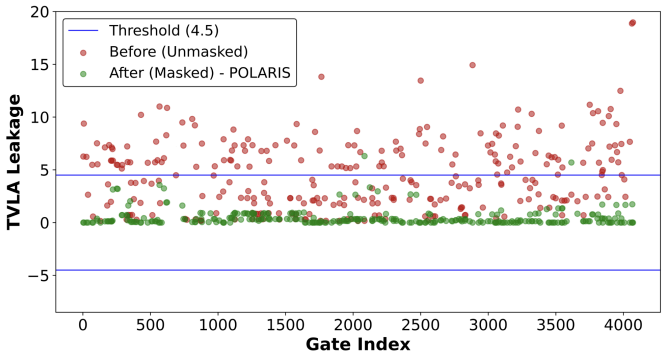


Fig. 4: TVLA values before and after masking in *des3* design. Gates exceeding threshold (±4.5) are considered as *leaky*.

POLARIS operates 6× faster than VALIANT, demonstrating its efficiency in leakage mitigation. Table IV compares area (μM^2), power (mW), and delay (ns) overheads of the unmasked design (Original) with the respective masked designs produced by VALIANT and POLARIS. As we can observe, POLARIS reduces area by 34.61%, power by 40.54%, and delay by 33.25% compared to VALIANT while also achieving greater leakage reduction. These results highlight the effectiveness of POLARIS in mitigating side-channel vulnerabilities with reduced resource overhead.

D. Explainability: POLARIS Extracted Rules

Using SHAP algorithms, we interpret the decision-making process for individual samples by examining waterfall plots in Fig. 3, which illustrate the contribution of each feature to the prediction of the target variable. Here, x represents a selected observation, $f(x)$ is the model's predicted value for this input, and $E[f(x)]$ denotes the expected value of the target variable, essentially the average prediction across all observations. The length of each bar in the plot indicates the SHAP value (Fig. 3). POLARIS can extract rules for leakage reduction; these rules are summarized in Table V.

E. Additional Analysis & Discussion

To visualize the effectiveness of POLARIS (see Fig. 4), we performed TVLA analysis for all gates in the *des3* design *before* (●) and *after* POLARIS *masking* (●). The performance of POLARIS can be improved by including a wider variety of designs in the training process, enabling POLARIS to better comprehend design diversity. POLARIS can also be extended to support other masking gates (e.g., DOM [5]), providing a flexible and robust solution for addressing power side-channel leakage mitigation with improved performance.

VI. Conclusion

Power side-channel leakage of sensitive on-chip data is a major security threat. Existing mitigation frameworks face scalability issues (limited data, TVLA reliance, slow runtime) and lack effectiveness. This paper presents the POLARIS, a DFS framework that can be integrated into commercial ASIC design flow to generate design-specific rules using XAI for low-overhead power side-channel mitigation. POLARIS is fast and utilizes a synthetic data generation scheme to bypass data-related concerns associated with most AI frameworks. We observe significant performance gain over state-of-the-art solutions in terms of speed, leakage reduction, and design overhead. Future works will make the POLARIS framework more efficient and extend it to other side-channels (EM/timing).

VII. Acknowledgment

This material is supported by the National Science Foundation (NSF) under Grant No. 2350363 and Grant No. 2316399.

REFERENCES

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[2] A. Moradi, "Side-Channel Leakage through Static Power – Should We Care about in Practice?" Cryptology ePrint Archive, Paper 2014/025, 2014. [Online]. Available: https://eprint.iacr.org/2014/025

[3] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side channel resistance validation," in *NIST Non-Invasive Attack Testing Workshop*, vol. 7, 2011, pp. 115–136.

[4] E. Trichina, "Combinational Logic Design for AES SubByte Transformation on Masked Data," Cryptology ePrint Archive, Paper 2003/236, 2003. [Online]. Available: https://eprint.iacr.org/2003/236

[5] H. Gross, S. Mangard, and T. Korak, "Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order," in *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, ser. TIS '16, New York, NY, USA, 2016, p. 3.

[6] R. Sadhukhan, S. Saha, S. Paria, S. Bhunia, and D. Mukhopadhyay, "VALIANT: An EDA Flow for Side-Channel Leakage Evaluation and Tailored Protection," *IEEE Transactions on Computers*, vol. 73, no. 2, pp. 436–450, 2024.

[7] D. Šijačić, J. Balasch, B. Yang *et al.*, "Towards efficient and automated side-channel evaluations at design time," *J Cryptogr Eng*, vol. 10, pp. 305–319, 2020.

[8] P. Slpsk, P. K. Vairam, C. Rebeiro, and V. Kamakoti, "Karna: A Gate-Sizing based Security Aware EDA Flow for Improved Power Side-Channel Attack Protection," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[9] T. Moos, F. Wegener, and A. Moradi, "DL-LA: Deep learning leakage assessment: A modern roadmap for SCA evaluations," Cryptology ePrint Archive, Paper 2019/505, 2019. [Online]. Available: https://eprint.iacr.org/2019/505

[10] M. Nair, R. Sadhukhan, H. Pearce, D. Mukhopadhyay, and R. Karri, "Netlist whisperer: Ai and nlp fight circuit leakage!" in *Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security*, ser. ASHES '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 83–92. [Online]. Available: https://doi.org/10.1145/3605769.3623989

[11] T. Schneider and A. Moradi, "Leakage Assessment Methodology - a clear roadmap for side-channel evaluations," Cryptology ePrint Archive, Paper 2015/207, 2015. [Online]. Available: https://eprint.iacr.org/2015/207

[12] O. Reparaz, "Detecting Flawed Masking Schemes with Leakage Detection Tests," in *Revised Selected Papers of the 23rd International Conference on Fast Software Encryption - Volume 9783*, ser. FSE 2016. Berlin, Heidelberg: Springer-Verlag, 2016, p. 204–222.

[13] E. Prouff and M. Rivain, "Masking against side-channel attacks: A formal security proof," in *Advances in Cryptology – EUROCRYPT 2013*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 142–159.

[14] P. Pathmanathan, S. Chakraborty, X. Liu, Y. Liang, and F. Huang, "Is poisoning a real threat to llm alignment? maybe more so than you think," *arXiv preprint arXiv:2406.12091*, 2024.

[15] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *Computer security–ESORICs 2020: 25th European symposium on research in computer security, ESORICs 2020, guildford, UK, September 14–18, 2020, proceedings, part i 25*. Springer, 2020, pp. 480–501.

[16] W. Guo, B. Tondi, and M. Barni, "An overview of backdoor attacks against deep neural networks and possible defences," *IEEE Open Journal of Signal Processing*, vol. 3, pp. 261–287, 2022.

[17] A. Turner, D. Tsipras, and A. Madry, "Label-consistent backdoor attacks," *arXiv preprint arXiv:1912.02771*, 2019.

[18] S. Lundberg, "A unified approach to interpreting model predictions," *arXiv preprint arXiv:1705.07874*, 2017.

[19] P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine learning guided structural analysis attack on hardware obfuscation," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2018, pp. 56–61.

[20] P. Chakraborty, J. Cruz, R. Almawzan, T. Mahfuz, and S. Bhunia, "Learning Your Lock: Exploiting Structural Vulnerabilities in Logic Locking," *IEEE Design & Test*, 2024.

[21] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you? Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[22] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan *et al.*, "Captum: A unified and generic model interpretability library for pytorch," *arXiv preprint arXiv:2009.07896*, 2020.

[23] J. Hayes, "ISCAS Benchmark Circuits for Logic Synthesis and Verification," accessed: 2024-11. [Online]. Available: https://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html

[24] EPFL-LSI, "The EPFL Combinational Benchmark Suite," accessed: 2024-11. [Online]. Available: https://www.epfl.ch/labs/lsi/page-102566-en-html/benchmarks/

[25] MIT-LL, "CEP: Common Evaluation Platform," accessed: 2024-11. [Online]. Available: https://github.com/mit-ll/CEP/

[26] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

CVMAX: Accelerator Architecture with Polar Form Multiplication for Complex-Valued Neural Networks

Hyunwuk Lee, Sungbin Kim, Sungwoo Kim, Won Woo Ro
School of Electrical and Electronic Engineering
Yonsei University, South Korea
{*hyunwuk.lee, sungbin.kim, sungwoo.kim, wro*}*@yonsei.ac.kr*

***Abstract*—Complex-Valued Neural Networks (CVNNs) have demonstrated high performance in applications where complex numbers are essential, but suffer from higher computational and memory overheads. Since their target applications often operate in resource-constrained environments, optimizing CVNNs for energy and area efficiency is important for their acceleration. To resolve these challenges, we present CVMAX, a software-hardware co-design for energy and area-efficient CVNN acceleration. CVMAX introduces a specialized quantization technique based on polar form representation and shift quantization. The technique significantly reduces the bit width of CVNNs and computational complexity compared to conventional quantization with rectangular form. Moreover, shift quantization leverages the computational simplicity of multiplication in polar form, reducing the complexity of complex number multiplication. With the quantization technique, we designed a dedicated hardware accelerator that supports CVMAX data and its associated arithmetic operations. In our evaluation, CVMAX achieves a 75% reduction in energy consumption and achieves a 4.44× speedup compared to conventional accelerators.**

1. Introduction

Complex Valued Neural Networks (CVNNs) are designed to effectively interact with applications that require complex numbers in their data. By leveraging the unique properties of complex numbers, CVNNs outperform conventional Real Valued Neural Networks (RVNNs) in tasks such as signal processing, MRI, and RADAR [1], [2], [3], [4], [5]. However, this advantage incurs significant computational and memory overhead. The arithmetic of complex numbers is inherently more computationally intensive, posing challenges for CVNNs [6]. Furthermore, target applications of CVNNs often run in resource-constrained environments, such as mobile devices, which require high energy and area efficiency to accelerate CVNNs in the hardware.

To address these challenges, we focus on polar form representation of complex numbers, a mathematical format for complex numbers. In polar form, a complex number is represented as $r \times e^{i\theta}$, where r denotes the magnitude and θ represents the phasor. Compared to the rectangular form $(a + bi)$, polar form offers two distinct characteristics. First, magnitude and phasor have constrained value ranges; magnitude is always positive and the phasor falls within the range of 0 to 2π. These limited value ranges enable more efficient quantization, reducing errors and improving compression rates [6]. However, prior work has only applied polar form quantization to memory optimization, without leveraging it for computation.

Second, a multiplication in polar form involves a simpler computation than in the rectangular form. In rectangular form, multiplication between two complex numbers requires four multiply, one add, and one subtract operation. In contrast, polar form only needs one multiply and one add operation to complete complex number multiplication. Despite the lower computational overhead of polar form in multiplication, polar form exhibits a disadvantage in addition. For the addition process, polar form necessitates conversion to the rectangular form. As a result, polar form and rectangular form have their own trade-off in their arithmetic.

Building on these characteristics, we propose CVMAX: an efficient accelerator architecture for Complex-Valued Neural Networks, incorporating polar form multiplication and shift quantization. CVMAX introduces the special co-design of the quantization and the hardware for efficient conversion between rectangular and polar forms. In CVMAX, we employ specialized shift quantization for the magnitude data of polar form, eliminating the need for the multiply operation in conversion process and complex number arithmetic. While conventional shift quantization has drawbacks in complex addition operations, we find that applying shift quantization solely to magnitude data avoids these issues and enables replacing multiplication with shift or add operations in complex number multiplication and conversion. With the quantization, we also design an accelerator architecture that performs complex number multiplication only with shifter and adder logic then utilize adder tree for complex number addition.

We summarize the contributions of CVMAX as follows:

- CVMAX leverages polar form quantization, capitalizing on its advantageous value range. For efficient complex number arithmetic, we adopt shift quantization for the magnitude part of the data.
- The CVMAX accelerator exploits simple polar form multiplication and a rectangular form adder tree in its processing element (PE) design. With CVMAX quantization, the PE of the CVMAX accelerator replaces all multipliers with shifters and adders, reducing both area and energy consumption.
- With CVMAX quantization, the CVMAX accelerator reduces PE area by 74% compared to conventional RVNN

Methods	Bit-width	Accuracy	Error
Baseline (Rectangular Form)	A 16/16 W 16/16	92.26%	-
Rectangular Form Quantization	A 8/8 W 8/8	90.98%	1.28%
Rectangular Form Quantization	A 8/8 W 4/4	90.40%	1.86%
Rectangular Form Quantization	A 4/4 W 4/4	84.28%	7.98%
Polar Form Quantization	A 8/8 W 8/8	91.44%	0.82%
Polar Form Quantization	A 8/8 W 4/4	90.81%	1.45%
Polar Form Quantization	A 4/4 W 4/4	88.12%	4.14%

Figure 1: Inference results of a CVNN model [13] applying uniform quantization in rectangular and polar forms at various bit widths. A refers to activation and W refers to weight. As complex number data is consists of two data, we denote bit width of each data in activation and weight.

accelerators. In an iso-area comparison, the CVMAX accelerator achieves a 4.44× speedup and 75% lower energy consumption compared to other accelerators.

2. Background and Motivation

A CVNN is a neural network that processes data as complex numbers and applies complex arithmetic in its layer operations [7], [8]. In certain applications, including MRI, RADAR, and sensor data processing, data is inherently represented as complex numbers due to theoretical foundations in those domains [9], [10]. In these areas, conventional RVNNs that operate only with real numbers would lose the information in the imaginary component and result in higher losses, although they also handle imaginary data as real numbers [10], [11], [12]. To address this, CVNNs are proposed to handle both the real and imaginary components by using complex numbers for their parameters and applying complex arithmetic in their layer operations and activation functions. As a result, CVNNs achieve higher performance in applications requiring complex-valued data [12]. However, CVNNs incur significant memory and computational overhead compared to RVNNs due to the increased complexity of handling complex-valued parameters and performing complex arithmetic [6].

Complex numbers are typically represented in two formats: rectangular form and polar form. In the rectangular form, a complex number is expressed as $a + bi$, where a and b are the real and imaginary components, respectively. In contrast, polar form represents a complex number using its magnitude and phase, expressed as $r \times e^{i\theta}$, where r is the magnitude and θ is the phasor. In the rectangular form, it does not have value limitation, so real and imaginary number can range from negative infinity to positive infinity. Unlike the rectangular form, polar form constrains values such that the magnitude is always positive, and the phasor ranges from 0 to 2π radian. Although both forms represent the same data, the narrower value range of polar form provides certain advantages, particularly in quantization.

Previous work [6] demonstrated that the narrow value range of polar form makes it more advantageous for quantization compared to rectangular form. The performance of quantization depends heavily on factors such as bit width, value range, and value distribution. Due to its narrower value range, polar form can reduce the required bit width or

Matrix Multiplication

$$c_{real} = \sum^{n} (a_{real} \times b_{real} - a_{img} \times b_{img})$$

Complex Number Multiplication → *4 multiply + 2 add / subtract*

$$c_{img} = \sum^{n} (a_{real} \times b_{img} + a_{img} \times b_{real})$$

Complex Number Addition → *2 add*

Total

$N \times$ **Complex Number Multiplication** $+ (N-1) \times$ **Complex Number Addition**

$= N \times (4\ \boldsymbol{multiply} + 2\ \boldsymbol{add/subtract}) + (N-1) \times 2\ \boldsymbol{add}$

(a) Rectangular Form

Matrix Multiplication

$$c_m e^{ic_\theta} = \sum^{n} a_m \times b_m e^{i(a_\theta + b_\theta)}$$

Complex Number Multiplication → *1 multiply + 1 add*

$$= \sum^{n} d_m e^{id_\theta}$$

Complex Number Conversion → *2 multiply + 2 sine/cosine*

Conversion

$$d_{real} = d_m \times \cos(d_\theta)$$

$$d_{img} = d_m \times \sin(d_\theta)$$

Complex Number Addition → *2 add*

Total

$N \times$ **(Complex Number Multiplication + Complex Number Conversion)**

$+ (N-1) \times$ **Complex Number Addition**

$= N \times (3\ \boldsymbol{multiply} + 1\ \boldsymbol{add} + 2\ \boldsymbol{sine/cosine}) + (N-1) \times 2\ \boldsymbol{add}$

(b) Polar Form

Figure 2: Arithmetic operations for computing a single output element in complex matrix multiplication using (a) rectangular form and (b) polar form. For simplicity, we derive the operations only for a single output element.

achieve lower quantization loss than rectangular form. However, prior work only applied quantization to store weights in memory and performed layer operations using decompressed weights and activations. To evaluate the potential of extending quantization to actual layer operations, we conducted an initial analysis of accuracy, as shown in Fig. 1. In this analysis, we applied basic symmetric quantization to both activations and weights of a CVNN model [13] in rectangular and polar forms at three different bit widths. The results show that quantization in polar form consistently achieves higher accuracy across all bit widths and retains accuracy at lower bit widths compared to rectangular form. With the analysis, we can verify that quantization with polar form is effective not only for weights but also for activations.

Since complex number matrix multiplication is the primary computational bottleneck in CVNNs, we also analyzed how arithmetic operations differ between the two formats. Fig. 2a shows the total arithmetic operations of two representation formats for a single output element of a complex number matrix multiplication. As depicted in Fig. 2a, the rectangular form requires two add operations for its add operation between complex numbers (e.g., $(a+bi)+(c+di) = (a+c)+(b+d)i$). However, its multiply operation requires four multiply, one add, and one subtract. In consequence, a single output element of N by N requires total $N \times (4\ multiply + 2\ add/subtract) + (N-1) \times 2\ add$.

In contrast, polar form exhibits different characteristics in their arithmetic. Unlike the rectangular form, an addition in polar form involves a more complex process that requires a conversion to the rectangular form. Fig. 2b shows that an addition in polar form involves two trigonometric operations and two multiply operations for conversion. After

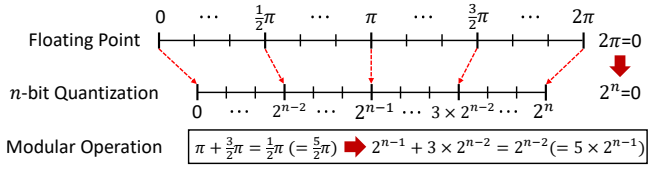


Figure 3: CVMAX quantization for phasor number with modular characteristics.

conversion, polar form addition proceeds in the same way as in rectangular form. To avoid the conversion, the previous work [6] adopts rectangular form for matrix multiplication, even though it uses polar form for the quantization.

However, we find that polar form is advantageous for multiplication. Polar form multiplication operations require only one multiply and one add operation. With this simpler multiplication process, polar form require $N \times (3\ multiply + 1\ add + 1\ sine + 1\ cosine) + (N-1) \times\ 2\ add$ for the same matrix multiplication. Considering that trigonometric operations can be simplified in hardware as in previous work [6], polar form consumes fewer multiply operation than rectangular form, even with the conversion.

3. CVMAX

With the analysis, we propose CVMAX: an efficient accelerator architecture with polar form quantization and multiplication for Complex-Valued Neural Networks. CVMAX targets polar-form matrix multiplication and introduces a co-designed specialized quantization technique and accelerator architecture. To leverage the simpler multiplication inherent to polar form, the CVMAX accelerator executes complex number multiplications using quantized polar-form data. We design the accelerator with an adder tree architecture, which can concurrently converts the data into rectangular form and sums the results. For a fast and lightweight matrix multiplication, CVMAX introduces specialized shift quantization, which applies shift quantization to the magnitude data of activations and weights. This approach eliminates all multiplication operations, replacing them with shift operations. Consequently, these techniques enable the CVMAX accelerator to replace multipliers with shifters and adders in its PE design, leading to significant reductions in both area and energy consumption. Moreover, the CVMAX quantization method, which leverages the benefits of polar form to reduce quantization error and loss, achieves superior performance compared to rectangular-form quantization. In the following subsections, we provide a detailed description of the CVMAX quantization method and the architectural design of the CVMAX accelerator.

3.1. CVMAX Quantization

To exploit the advantages of polar form in quantization, CVMAX quantization applies specialized quantization to the magnitude and phasor data. For phasor data, we extend the quantization method from previous work [6] by leveraging the modular characteristics of the phasor, as shown in Fig. 3. Unlike conventional uniform symmetric quantization, we define the value range of phasor as 0 to 2π with $2^{bitwidth}+1$

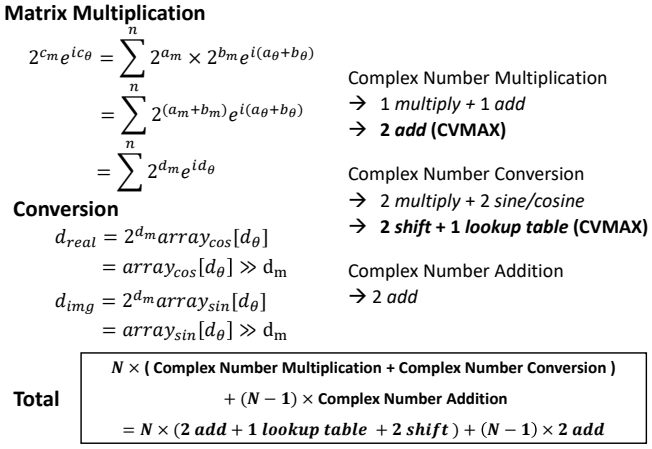


Figure 4: CVMAX matrix multiplication.

bins rather than $2^{bitwidth}$. This modification accounts for the circular nature of the phasor, where 2π is equivalent to 0. By using $2^{bitwidth}+1$ bins, CVMAX can effectively represent special values within the phasor, as illustrated in the figure. Since the outputs of trigonometric functions are zero for these special values, representing those numbers in quantization is critical for accuracy. While conventional quantization with $2^{bitwidth}$ bins fails to capture all these special values with uniform quantization, CVMAX quantization can reduce the bit width required for phasor data while preserving accuracy by leveraging this modular design.

In complex number matrix multiplication, the magnitude involves multiply operations between magnitudes and trigonometric values, as shown in Fig. 2b. To eliminate these multiply operations, we focus on shift quantization [14], which replaces multiplications with shift operations. In RVNNs or rectangular form data, applying shift quantization to both weight and activation often leads to high accuracy loss and complicated addition between shift-quantized data. However, we focus on that magnitude numbers do not necessitate addition operations with each other during complex number matrix multiplication. With this characteristic, we find out that applying shift quantization exclusively to the magnitude of both weights and activations avoids these challenges. Moreover, since the magnitude is always a positive number, shift quantization in CVMAX does not require a sign bit for negative values, resulting in further bit-width savings. With the observation, CVMAX employs shift quantization for the magnitude of both weights and activations, effectively replacing multiply operations between magnitudes with add operations and multiply operation with the results of trigonometric functions with shift operations, as shown in Fig. 4.

With this specialized quantization, CVMAX changes the arithmetic operations required for complex matrix multiplication, as illustrated in Fig. 4. For each complex number multiplication, CVMAX requires two add operations: one for adding the phasors and another for combining the magnitudes. Since magnitudes in CVMAX are represented as exponents due to shift quantization, the multiplication

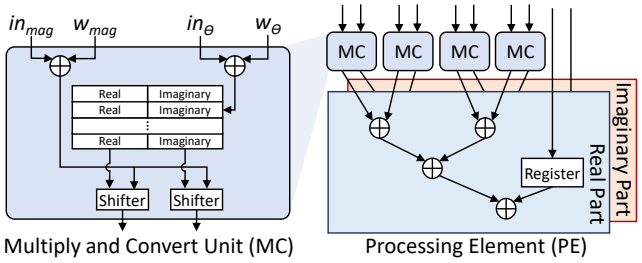


Figure 5: Single PE design of the CVMAX accelerator. The PE comprises a Multiply and Convert (MC) unit and a coupled adder tree unit.

of two magnitudes is transformed into an addition of their exponents. The conversion process involves a lookup table search and two shift operations. CVMAX adopts the lookup table optimization from previous work [6] for trigonometric operations, enabling the use of a single dual-output lookup table. This approach replaces complex operations with one lookup table search and minimizes computation overhead. Multiplications between the results of trigonometric functions and magnitudes are also converted into shift operations using CVMAX quantization. As a result, CVMAX involves total $N \times (2\ add + 1\ lookup\ table\ search + 2\ shift) + (N - 1) \times\ 2\ add$ for a single output element of matrix multiplication. Since multiplications are computationally expensive than additions or shift operations, CVMAX effectively reduces computational complexity of CVNN layers.

In addition, CVMAX also designs the quantization for rectangular form, as complex number addition is performed in rectangular form. In CVMAX, the rectangular form is used for representing the result of trigonometric functions, ranging from -1 to 1, and multiplication results between the results and magnitude. Consequently, the value ranges of these results are exactly twice as large as the value range of magnitudes. CVMAX offers the flexibility to assign different bit widths to rectangular-form data compared to phasor data by configuring the bit width of the stored results of trigonometric functions. In our default configuration, the bit width of the stored results is set at 8-bit, while the bit widths for both phasor and magnitude data are set to 4 bits. Using a 4-bit magnitude allows up to 8 bits of shift operations to be applied to an 8-bit result of trigonometric function, resulting in 16-bit rectangular-form data for the addition process. By allocating more bit width to the results of trigonometric functions and addition operations, CVMAX reduces the size of polar-form data, which is critical for minimizing the area and energy consumption of the accelerator. This design strategy enables CVMAX quantization to maintain accuracy comparable to 8-bit quantization in other methods, while employing only 4-bit magnitudes and 4-bit phasors.

3.2. CVMAX Accelerator

The CVMAX accelerator hardware is specifically designed to take full advantage of the optimizations introduced by CVMAX quantization, particularly through the use of shift operations and the elimination of multipliers. The accelerator consists of adder tree-based PEs and activation units. The CVMAX accelerator executes CVNN layer operations using the specialized PEs, while activation functions are handled by the activation unit. The PE features a specialized adder tree architecture that follows the CVMAX matrix multiplication process.

Fig. 5 illustrates the detailed design of a single PE in the CVMAX accelerator. The PE is implemented as an adder-tree-based multiply and accumulate (MAC) unit, consisting of two key components: the Multiply and Convert (multiply and convert (MC)) unit and the adder tree. The MC unit performs complex number multiplication using CVMAX data by applying polar-form multiplication and then converting the data into rectangular form. Since CVMAX quantization replaces multiplication between magnitudes with addition, the MC is composed of two adders, two shifters, and one dual-output lookup table, eliminating the need for multipliers. For complex number multiplication, the MC unit adds the magnitudes and phasors from the inputs and weights, as shown in Fig. 4. Using the added phasor, the MC probes precomputed value of trigonometric functions in the lookup table and retrieves two outputs corresponding to the real and imaginary components. Finally, the MC applies shift operations to both data using the added magnitude. Through these operations, the MC performs complex number multiplication in polar form and converts the data into rectangular form.

The adder tree within the PE handles the complex number additions in rectangular form using the outputs generated by the MCs. To sum the real and imaginary components of the complex numbers, the adder tree is designed as a coupled structure: one part processes the real components, while the other processes the imaginary components. Each adder tree receives its respective inputs from the MCs and computes the partial sums for each component. This coupled design allows the adder tree to produce two outputs, representing the partial sums of the real and imaginary components.

In CVMAX quantization, the default bit width for both the magnitude and phasor is 4 bits. As a result, the CVMAX accelerator employs 4-bit adders in the MC unit along with a 16-entry lookup table. To accommodate with the 8-bit trigonometric value, the lookup table is configured with an 8-bit width, while the adder tree operates with corresponding bit widths for its outputs. This configuration optimizes the hardware utilization by allocating higher bit widths to certain logic while keeping the phasor and magnitude at lower bit widths. This design minimizes hardware overhead while preserving the accuracy of CVMAX. By replacing traditional multipliers with lookup tables, adders, and shifters in the MC unit, the CVMAX accelerator achieves single-cycle operation for the PE. This reduces the number of cycles required for matrix multiplication compared to conventional rectangular-form implementations.

While the PE unit accelerates complex number matrix multiplications in CVNN layers, the activation unit handles other operations required by CVNNs. First, the activation unit computes activation functions specific to CVNNs (e.g., modRelu, zRelu, and cardioid). Additionally, it performs conversions between polar and rectangular forms. This conversion process is integrated with requantization, a critical

TABLE 1: CVNN Benchmarks for Quantization

Model	Tasks	Dataset
CVNN_1 [18]	Image Generation	MRI Data [19]
CVNN_2 [13]	PolSAR classification	PolSF [20]

step for accelerators that use quantization. By combining the conversion and requantization processes, the CVMAX activation unit eliminates the need for separate operations, streamlining overall execution.

The CVMAX accelerator dynamically determines the sequence of conversion and activation function execution based on the required format by the activation function. For instance, if the activation function requires polar-form data, the activation unit first performs the conversion process before applying the activation function. Conversely, if the activation function operates in rectangular form, the unit executes the activation function first and then converts the data into polar form for subsequent layer operations. Since these two operations take fewer cycles than matrix multiplication executed on the PE units, it operates seamlessly within the pipelined architecture, avoiding stall cycles.

4. Evaluation

4.1. Methodology

To evaluate the quantization of CVMAX, we implemented CVMAX quantization in two CVNN models using the PyTorch [15] and TensorFlow [16] frameworks. Table 1 lists the CVNN models and their corresponding datasets used in our evaluation. Each model was evaluated on a specific task, with performance metrics tailored to the nature of the task. For CVNN_1 [17], an image generation task, the performance metric used is Structural Similarity Index Measure (SSIM), where higher values indicate better performance. In contrast, for CVNN_2 [13], a classification task, we used accuracy. We adopted the dataset and evaluation metric from original papers [13], [17].

We conducted a comparative evaluation of the CVMAX accelerator against other RVNN accelerators using a cycle-accurate simulator. To compare with various types of accelerators, we evaluated two RVNN accelerator designs: a systolic-array-based accelerator and an adder-tree-based accelerator. For the systolic-array-based accelerator, we selected a TPU-like [21] model based on an open-source simulator [22]. For the adder-tree-based accelerator, we used a tensor core-like model based on Accel-Sim [23]. Additionally, we evaluated the accelerator from previous work [6], which is a systolic-array-based design targeting CVNNs. For executing CVNN on RVNN accelerator, we assume that each CVNN layer is split to four sub-layer as described in previous work [6]. To measure the area and energy consumption of the PE and activation unit, we used the Synopsys Design Compiler with the FreePDK 45nm library and CACTI [24].

4.2. Evaluation on Quantization

To assess the performance of quantization, we set the baseline models to use FP16 for their real and imaginary data and evaluated them using four quantization methods. The first two quantization methods are basic quantization methods applied in rectangular and polar form, as described in Section 2. The remaining two are specialized quantization methods designed specifically for CVNNs. The first of these is PAQ [6], a prior work that applies quantization in polar form. While the original implementation only applies quantization to weight data, we extended it to include both activations and weights for a fair comparison with other methods. The final method is the proposed CVMAX quantization. In our evaluation, we considered two bit-width configurations for the quantization methods. We applied 8-bit and 4-bit for each data to the quantization methods, with 16-bit and 8-bit used for addition operations, respectively. For CVMAX quantization, we introduced an additional bit-width configuration: a 4-bit magnitude combined with an 8-bit phasor. This configuration was chosen because a 4-bit shift-quantized magnitude in CVMAX can represent shifts ranging from 8-bit shift right to an 8-bit shift left, effectively generating 16-bit data for the addition process.

Quantization Methods	Bit-width	CVNN_1		CVNN_2	
		SSIM	Error	Accuracy	Error
Baseline	A16/16 W16/16	0.910	-	92.26%	-
Rectangular From	A8/8 W8/8	0.891	0.019	90.98%	1.28%
Rectangular From	A4/4 W4/4	0.821	0.089	84.28%	7.98%
Polar From	A8/8 W8/8	0.897	0.013	91.44%	0.82%
Polar From	A4/4 W4/4	0.855	0.055	88.12%	4.14%
PAQ	A8/8 W8/8	0.899	0.011	91.51%	0.75%
PAQ	A4/4 W4/4	0.860	0.050	89.99%	2.27%
CVMAX	A8/8 W8/8	0.898	0.012	91.49%	0.77%
CVMAX	A4/8 W4/8	0.890	0.020	91.01%	1.25%
CVMAX	A4/4 W4/4	0.871	0.039	90.00%	2.26%

Figure 6: Inference results of CVNN models [13], [18] applying various quantization methods across various bit widths. A refers to activation and W refers to weight.

Fig. 6 provides the inference results of our benchmarks with the above quantization methods. As shown in the figure, the CVMAX quantization design with 4-bit for each data outperforms the other quantization methods using 4-bit bit-width. These results stem from the characteristics of shift quantization, which benefits from covering a larger value range with fewer bits. As a result, the CVMAX quantization design maintains accuracy within a 1% error margin compared to 8-bit rectangular form quantization. Besides, as the shift quantization exhibit poor accuracy preserving in larger bit widths, CVMAX shows lower accuracy than PAQ in the design with 8-bit for each data. Furthermore, CVMAX quantization with 4-bit magnitude and 8-bit phasor achieves comparable performance with other quantization of 8-bit, while reducing data size by 25% through the reduction of the magnitude to 4-bit. These results highlight the effectiveness of CVMAX quantization in reducing bit width while maintaining an acceptable error rate.

4.3. Area Analysis

We compared the area of the CVMAX accelerator with conventional accelerators. For a fair comparison, we standardized the computational capabilities of the accelerators to a 64 by 64 systolic array in a TPU-like architecture.

TABLE 2: Area Comparison

	Bit width	Relative Area
TPU-like	A8/8 W8/8	1.00
Tensor Core-like	A8/8 W8/8	0.78
PAQ Accelerator	A8/8 W8/8	1.02
CVMAX Accelerator	A4/4 W4/4	0.26

TABLE 3: Hardware configuration for iso-area comparison

	# PEs	# Multiplications
TPU-like	4096	4096
Tensor Core-like	1310	5240
PAQ Accelerator	4015	4015
CVMAX Accelerator	962	20960

Since one PE of the CVMAX accelerator can execute four complex number multiplications —which would require 16 multiplications in RVNN accelerators —we set the number of CVMAX PEs to 256. Similarly, we set the number of tensor cores to 1024, as each core executes four real number multiplications. Table 2 shows the results of our area comparison. The CVMAX accelerator achieves a 74% and 65% area reduction compared to the TPU-like accelerator and the tensor core-like design and 75% area reduction compared to the PAQ accelerator.

4.4. Performance and Energy Analysis

To ensure a fair comparison for performance and energy, we conducted the evaluations based on iso-area criteria across four distinct accelerator designs. The area configuration for performance evaluation is detailed in Table 3. According to Fig. 6, we adjusted the bit widths of the four accelerators to maintain accuracy close to that achieved with 8-bit rectangular form quantization, the quantization method used in TPU-like and tensor core accelerators. For the iso-area comparison, we set the accelerators to have the same area as a 64 by 64 TPU-like systolic array. We set the number of CVMAX PEs to 962, which can execute 20960 real number multiplications at once, and configured the other accelerators similarly. Furthermore, to broaden the scope of evaluation, we included two additional benchmarks for performance analysis, referencing virtual CVNN models generated from conventional RVNN models. For the models, we convert all data of models into complex number and layers to its counter part layer in CVNN.

Fig. 7 illustrates the relative performance of CVNNs in various accelerator architectures. Notably, the CVMAX accelerator demonstrates a 4.44× and 3.9× speedup compared to the TPU-like model and the tensor core-like accelerator, and 3.1× speedup compared to the PAQ accelerator, respectively. These performance gains come from the larger computational capabilities of the CVMAX accelerator which can execute about five times more multiplications than TPU-like models at once, as shown in Table 3.

We also evaluated the energy consumption of the accelerators, as shown in Fig. 8. The CVMAX accelerator reduces energy consumption by 75% and 73% compared to the TPU-like model and the tensor core-like accelerator, and 65% energy consumption than the PAQ accelerator, respectively. The energy savings of CVMAX are mainly achieved with

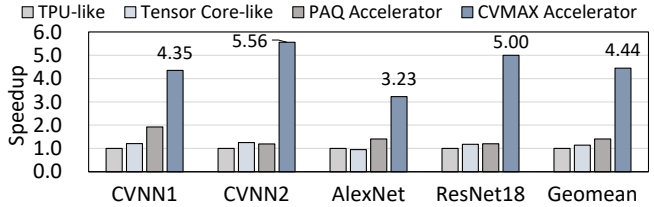


Figure 7: Relative performance of CVNNs in various accelerator architectures. All performance results are normalized to TPU-like model.

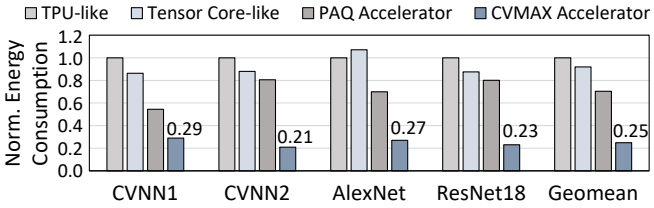


Figure 8: Relative energy consumption of CVNNs in various accelerator architectures.

reduced cycle times and reduced buffer and memory usage. In the evaluation, we adopted the same memory and buffer sizes for all accelerators. However, resizing the memory and buffer according to the reduced bit width could result in additional energy savings by lowering the static power of the memory and buffer and reducing the total area of the accelerator.

5. Conclusion

CVNNs exhibit high performance in applications that use complex numbers, but also incur significant computational and memory overheads due to their complex-valued data and layer operations. To mitigate these overheads, we propose CVMAX, a software-hardware co-design for energy- and area-efficient acceleration of CVNNs. CVMAX employs quantization techniques that effectively reduce bit widths and replace multiply operations with shift and add operations by adopting power-of-two-based shift quantization for magnitudes. This specialized quantization technique enables the CVMAX accelerator to support CVNN layer operations without a multiplier, which typically consumes high energy and occupies substantial area in the hardware budget. By eliminating the multiplier, CVMAX achieves a 75% reduction in energy consumption and a 4.44× speedup in iso-area comparisons, while maintaining an acceptable error rate compared to conventional hardware and software designs.

Acknowledgments

This work was supported by Samsung Electronics Co., Ltd(IO201210-07936-01, 50%) and Institute of Information & communication Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2025-0-00441, Memory-Centric Architecture Using the Reconfigurable PIM Devices, 50%). Won Woo Ro is the corresponding author.

References

[1] C.-A. Popa, "Complex-valued convolutional neural networks for real-valued image classification," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 816–822.

[2] M. Wilmanski, C. Kreucher, and A. Hero, "Complex input convolutional neural networks for wide angle sar atr," in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2016, pp. 1037–1041.

[3] M. Scarpiniti, D. Vigliano, R. Parisi, and A. Uncini, "Generalized splitting functions for blind separation of complex signals," *Neurocomputing*, vol. 71, no. 10-12, pp. 2245–2270, 2008.

[4] X. Wang, P. Chen, H. Xie, and G. Cui, "Through-wall human activity classification using complex-valued convolutional neural network," in *2021 IEEE Radar Conference (RadarConf21)*. IEEE, 2021, pp. 1–4.

[5] J. A. Barrachina, C. Ren, C. Morisseau, G. Vieillard, and J.-P. Ovarlez, "Complex-valued vs. real-valued neural networks for classification perspectives: An example on non-circular data," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 2990–2994.

[6] H. Lee, H. Jang, S. Kim, S. Kim, W. Cho, and W. W. Ro, "Exploiting inherent properties of complex numbers for accelerating complex valued neural networks," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. Association for Computing Machinery, 2023, p. 1121–1134.

[7] C. Lee, H. Hasegawa, and S. Gao, "Complex-valued neural networks: A comprehensive survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 8, pp. 1406–1426, 2022.

[8] J. Bassey, L. Qian, and X. Li, "A survey of complex-valued neural networks," *arXiv preprint arXiv:2101.12249*, 2021.

[9] Y. Sunaga, R. Natsuaki, and A. Hirose, "Land form classification and similar land-shape discovery by using complex-valued convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 10, pp. 7907–7917, 2019.

[10] A. M. Sarroff, "Complex neural networks for audio," Ph.D. dissertation, Dartmouth College, 2018.

[11] G. Shi, M. M. Shanechi, and P. Aarabi, "On the importance of phase in human speech recognition," *IEEE transactions on audio, speech, and language processing*, vol. 14, no. 5, pp. 1867–1874, 2006.

[12] P. Virtue, X. Y. Stella, and M. Lustig, "Better than real: Complex-valued neural nets for mri fingerprinting," in *2017 IEEE international conference on image processing (ICIP)*. IEEE, 2017, pp. 3953–3957.

[13] J. Barrachina, C. Ren, C. Morisseau, G. Vieillard, and J.-P. Ovarlez, "Complex-valued neural networks for polarimetric sar segmentation using pauli representation," in *IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2022, pp. 4984–4987.

[14] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li, "Deepshift: Towards multiplication-less neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 2359–2368.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *Osdi*, vol. 16, no. 2016. Savannah, GA, USA, 2016, pp. 265–283.

[17] M. A. Dedmari, S. Conjeti, S. Estrada, P. Ehses, T. Stöcker, and M. Reuter, "Complex fully convolutional neural networks for mr image reconstruction," in *International Workshop on Machine Learning for Medical Image Reconstruction*. Springer, 2018, pp. 30–38.

[18] E. Cole, J. Cheng, J. Pauly, and S. Vasanawala, "Analysis of deep complex-valued convolutional neural networks for mri reconstruction and phase-focused applications," *Magnetic resonance in medicine*, vol. 86, no. 2, pp. 1093–1109, 2021.

[19] F. Ong, S. Amin, S. Vasanawala, and M. Lustig, "Mridata. org: An open archive for sharing mri raw data," in *Proc. Intl. Soc. Mag. Reson. Med*, vol. 26, no. 1, 2018.

[20] X. Liu, L. Jiao, F. Liu, D. Zhang, and X. Tang, "Polsf: Polsar image datasets on san francisco," in *Intelligence Science IV: 5th IFIP TC 12 International Conference, ICIS 2022, Xi'an, China, October 28–31, 2022, Proceedings*. Springer, 2022, pp. 214–219.

[21] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1–14.

[22] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

[23] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 473–486.

[24] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

CREST-CiM: Cross-Coupling-Enhanced Differential STT-MRAM for Robust Computing-in-Memory in Binary Neural Networks

Imtiaz Ahmed, Akul Malhotra, Sumeet Kumar Gupta
Purdue University, West Lafayette, IN, USA
ahmed202@purdue.edu

***Abstract*—We propose CREST-CiM, an STT-MRAM-based Computing-in-Memory (CiM) technique, targeted for binary neural networks. To circumvent the low-distinguishability issue in standard MRAM-based CiM, CREST-CiM utilizes two magnetic tunnel junctions (MTJs) to store $+1$ and -1 weights in a bit-cell and cross-couples the MTJs, achieving a high-to-low current ratio of up to 8100 for a bit-cell. Our analysis for 64×64 arrays shows up to 3.4x higher CiM sense-margin, 27.6% higher read-disturb-margin, and resilience to process variations, and other hardware non-idealities, albeit at the cost of just 7.9% overall-area overhead, and <1% energy and latency overhead compared to a 2T-2MTJ-CiM design. Our system-level analysis for ResNet-18 trained on CIFAR-10 shows near-sofware inference accuracy with CREST-CiM, with 10.7% improvement over 2T-2MTJ baseline.**

***Index Terms*—Binary Neural Networks (BNNs), Compute-in-Memory (CiM), Cross-coupled, Spin-Transfer-Torque-Magnetic Random Access Memory (STT-MRAM), Tunneling Magnetoresistance (TMR), XNOR**

I. Introduction

Deep Neural Networks (DNNs) have revolutionized the landscape of artificial intelligence (AI) for a large number of tasks ranging from classification to generative text and images. Yet their implementation, especially on resource-limited edge devices, is hindered by their enormous storage and energy demands, stemming primarily from the growing size of the DNN models and a large number of memory-processor transactions [1]. To counter the latter, computing-in-memory (CiM) offers promise by performing matrix-vector multiplications (MVM) within the memory array, thereby alleviating energy-hungry data transfer between the memory and processor [2] [3].

To support the increasing number of DNN parameters, while achieving high energy efficiency, quantization of weights and activations to lower bit precision has been widely explored, which further assists CiM to meet the energy targets of edge devices. This strategy reduces the computational, storage, and communication costs with only a mild effect on accuracy [4]. Among these strategies, binary neural networks (BNNs) mark a significant leap forward towards ultra-low bit precision, encoding weights and activations with two values -1 and $+1$. This results in a drastic improvement in terms of energy and area efficiencies [5] while retaining acceptable inference accuracy for certain tasks [6].

Utilizing CiM in BNN accelerators (CiM-BNNs) yields synergistic improvements in the energy efficiency [7], [8]. Further improvements are possible with emerging non-volatile memories (NVMs), which address several limitations of standard memories. [3], [9]–[13]. Amongst various NVM technologies used for CiM-BNNs, spin-based technologies such as spin-transfer torque magnetic RAMs (STT MRAMs) are particularly advantageous in terms of offering non-volatility along with high endurance, and CMOS-compatible process and programming voltages [14]. However, one glaring issue that spin-memories suffer from is their low tunneling magnetoresistance (TMR) ratio of the magnetic tunnel junction (MTJ), which leads to poor distinguishability between the binary states. A maximum room temperature TMR of 604% has been reported in [15] and this may pave way for robust read operation in standard memories. However, for CiM (which involves assertion of multiple word-lines and sensing of multiple output states), this problem needs systematic investigation. The low TMR issue is further aggravated by hardware non-idealities such as parasitic resistances in CiM arrays and process variations [16].

Various efforts have been made to tackle non-idealities and the inherently poor distinguishability of MTJs,like device-algorithm co-design of spin-orbit-torque (SOT) MRAM-CiM [17] and others [11]. In the context of BNNs, specialized differential bit-cells have been proposed [18], [19], which utilize 2 MTJs per weight bit to store -1 and +1. Most of such designs trade-off area for a robust CiM and/or to cater to BNN-specific needs. In fact, CiM-driven optimizations sacrificing memory density for robustness have been utilized for other technologies as well [3].

Another common technique to reduce non-ideal impacts is partial wordline activation (PWA), in which only a subset of rows in an array are asserted in a single CiM cycle. This also reduces ADC costs, albeit at the cost of compute-latency [13]. Since PWA does not incur major area overheads, it has been used in conjunction with other non-ideality mitigation techniques, such as those discussed above and in [20]. Despite these optimizations, CiM robustness in MRAMs is still insufficient, making it important to explore new design approaches for MRAM-based CiM.

To improve the CiM robustness and inference accuracy of spin-based designs, in this paper, we propose an STT-MRAM-

based cross-coupled differential bit-cell and CiM-array aimed for BNNs. Referred to as CREST (cross-coupling-enhanced STT MRAM)-CiM, the proposed technique achieves a large boost in the ratio of high and low currents (I_H/I_L) of the bit-cell, which translates to robust CiM and high inference accuracy. The contributions of our work are as follows:

- We propose a CiM-optimized STT-MRAM bit-cell that stores signed binary weights {+1,-1} in two MTJs and cross-couples them to enhance the distinguishability. We show that the proposed CREST bit-cell achieves maximum $I_H/I_L \approx 8150$ and leads to a simultaneous improvement in the read disturb margin (RDM), albeit with some area penalty.
- We design a 64×64 CREST-CiM array to compute dot products of *signed* binary weights and inputs.
- We analyze the implications of circuit non-idealities including device-to-device variations and parasitic resistances of the array (such as wire resistance, loading resistance of periperal circuits), and show improvement in CiM robustness compared to previous spin-based designs.
- We analyze the inference accuracy of ResNet18 BNN trained on the CIFAR10 dataset. By incorporating the hardware attributes of CREST-CiM and non-idealities within the inference engine, we show higher inference accuracy for CREST-CiM compared to previous MRAM-CiM designs.

II. Background & Related Works

An MTJ consists of a pinned layer (PL) and a free layer (FL) of ferromagnets-separated by a tunneling oxide (MgO) with thickness t_{MgO} [3]. Binary weights are stored as anti-parallel (AP) (magnetization of FL and PL in opposite direction) or parallel (P) (in the same direction) states of the MTJ, with the former having higher resistance than the latter ($R_{AP} > R_P$). Achieving large ratio of R_{AP} and R_P or high TMR ($\frac{R_{AP}-R_P}{R_P} \times 100\%$) is a major challenge in MRAMs, which is even more critical for CiM, as discussed in this section.

A. CiM-BNN Designs

Before we discuss the challenges in MRAM-based CiM, let us present a brief overview of the design options for BNNs. Since the weights and activations in BNNs are in the {-1,+1} regime, the computation of the scalar product of weights and activations corresponds to XNOR operation (unlike standard CiM designs based on {0,1} weights/inputs and AND-based scalar product). There are two design strategies for enabling XNOR-CiM in BNNs. One is to design specialized differential bit-cells with 2 MTJs to store complementary -1 and $+1$ weights [18] and perform in-situ XNOR-CiM. The second approach is based on NANDNet design [7], which utilizes standard bit-cells along with transformation of inputs and outputs. While the first technique offers higher CiM robustness (due to the differential design - details later), the NANDNet design achieves XNOR-CiM at a lower array area. However, NANDNet requires extra peripheral circuitry to perform data transformations, including an adder tree for dynamically computing the sum of inputs [7]. This leads to mild hardware costs.

B. Challenges Due to Hardware Non-Idealities & Poor TMR

CiM designs based on MRAMs face several challenges due to circuit non-idealities (such as parasitic resistances and process variations) and low distinguishability of the binary states. Since CiM involves assertion of multiple rows, the current through the memory bit-lines can be much larger than a standard memory read. This leads to significant IR drops on the driver/wire resistances causing deviation of the output current from ideal case [16]. This results in computational errors and can result in a drop in inference accuracy. This problem is aggravated by the low TMR of the MTJ, which not only reduces distinguishability but also yields large current corresponding to 0 state (I_L). I_L from multiple cells can add up to produce a false non-zero output. Also, large I_L from multiple bit-cells aggravates the IR drops. This along with small ratio of high and low current (I_H/I_L), yields poor sense margins (SM), severely degrading the CiM robustness.

C. Approaches to Improve CiM Robustness in MRAMs

Techniques to improve CiM robustness include dummy column usage to address false non-zero outputs due to high I_L, which subtracts the undesired I_L accumulations [20]. However, this subtraction is not perfect due to different IR drops in the real and dummy columns. Utilizing differential bit-cells is another approach [18], which alleviates the effect of hardware non-idealities and interestingly, can also be designed for in-situ XNOR-CiM. To lower the IR drops, PWA has been used [13], as noted before. In addition to reducing the non-idealities, PWA has also been shown to lower the enormous costs of analog-to-digital converters (ADCs) but at the cost of latency. Other design approaches include utilizing bit-cells with decoupled compute-write paths [17], which allows for CiM-driven MTJ optimization, but at the cost of area compared to standard spin transfer torque (STT) MRAMs.

While improving the CiM robustness, these techniques do not address the primary issue that stem from large I_L produced by the bit-cell. As this challenge is detrimental to both CiM and standard memory operations, increasing MTJ TMR has been an active research pursuit, including utilizing correlated materials (CM) [21], voltage-boosting to amplify TMR [22] and dynamic latching in conjunction with standard 1T-1MTJ cell [23]. However, none of these memory-focused approaches have been explored in the context of CiM.

In this paper, we attack the fundamental problem of high I_L (arising at the bit-cell level) in the context of CiM targeted for BNN designs. By proposing a novel bit-cell that utilizes cross-coupling to reduce I_L and enhance I_H/I_L, we show a significant improvement in CiM robustness,translating to high system-level inference accuracy.

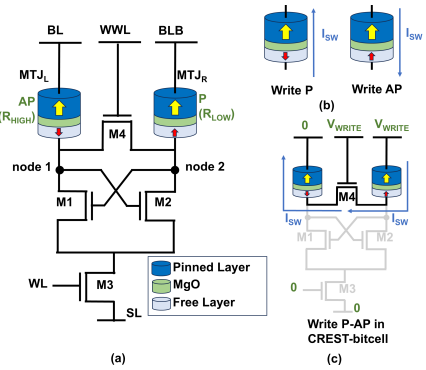


Fig. 1. Schematic and write operation of CREST-bitcell, 1(a) Schematic of proposed bit-cell, 1(b) Switching current direction for P (left) and AP (right) write, 1(c) Controlling P-AP write operation in CREST-bitcell with transistor M4.

TABLE I
DEVICE AND CIRCUIT PARAMETERS FOR HSPICE SIMULATIONS

Parameters	Value
Free Layer Dimension, $W \times L$ (nm^2)	60×60
FL Thickness, T_M (nm)	1
MgO Thickness, t_{ox} (nm)	1.3
Saturation Magnetization, M_s (emu/cm^3)	865 [24]
Uniaxial Anisotropy Density, K_u (erg/cm^3)	9.66×10^5 [25]
Energy Barrier, $E_B(k_BT)$	64
Damping Coefficient, α	0.008
Gyromagnetic Ratio, γ (MH_z/Oe)	17.6
Wire Resistance, $R_w(\Omega/\mu m)$	3.3 [26]
Driver Resistance, R_D (Ω)	250

D. Simulation Framework

For characterizing STT-MRAM bit-cell, dynamics of MTJ are captured using the Landau-Lifshitz-Gilbert-Slonczewski model to obtain the FL magnetization switching behavior [27]. We utilize the non-equilibrium Green's function (NEGF) to model MTJ resistance [27]. Experimentally calibrated parameters used for the perpendicular magnetic anisotropy (PMA) MTJ used in our simulations are listed in Table I. The MTJ TMR in our analysis is ∼400% to 500% following the works in [15], [17]. For the transistors, we employ the predictive technology models corresponding to the 45nm technology node [28], [29].

III. CREST-CiM: The Proposed Solution

In this section, we explain the write, read, and XNOR -based CiM operations of our proposed cross-coupled bit-cell.

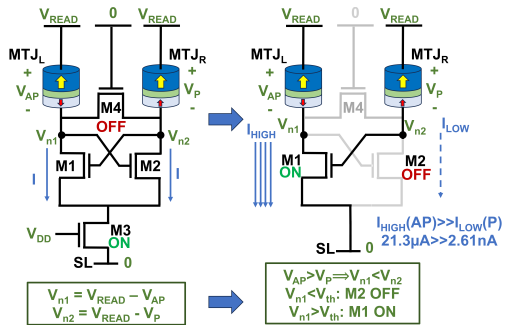


Fig. 2. Read operation of CREST-bitcell for stored weight AP-P. As $V_{READ} > V_{th}$, internal node voltage at AP branch (V_{n1}) is smaller than P branch (V_{n2}). This turns M2 transistor OFF, resulting in M1 transistor turning ON. Such cross-coupling action causes I_P reduction and I_{AP}/I_P ratio boost.

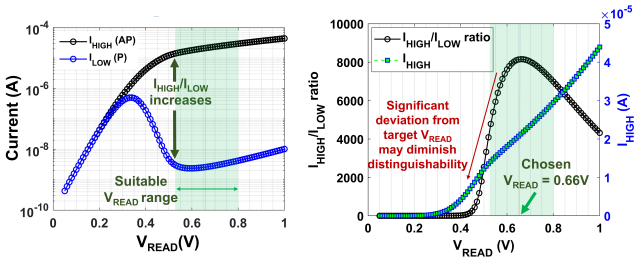


Fig. 3. Current vs V_{READ} (left) and I_{HIGH}/I_{LOW} vs V_{READ}(right) plots with safe operating voltage range marked.

A. Proposed CREST Bit-cell

The schematic of our cross-coupling-enhanced STT MRAM (CREST) bit-cell is shown in Fig. 1a. Our design features two complementary MTJs, MTJ_L and MTJ_R, to store the signed binary weights, (similar to previous 2T-2R designs [18]). MTJ_L in the parallel (P) state and MTJ_R in the anti-parallel (AP) state denote weight = +1, while AP in MTJ_L and P in MTJ_R encode weight = -1. The MTJs are cross-coupled via transistors M1 and M2, while a common access transistor M3 is controlled by the wordline (WL). Write access transistor M4 connects the two MTJs and is controlled by write wordline (WWL). PL of MTJ_L and MTJ_R are connected to the bit-line, BL and BLB, respectively. Let us now discuss the write and read operations.

1) Write Operation: To program $+1$ (MTJ_L:P; MTJ_R: AP), $V_{WRITE}(0)$ is applied on BLB(BL) and WWL is asserted, while WL is kept at 0 (Fig. 1c). This leads to current from BLB to BL switching MTJ_R to AP state (due to current from PL to FL, Fig. 1b) and MTJ_L to the P state (due to current from FL to PL). Similarly, to program -1 (MTJ_L:AP; MTJ_R: P), the current direction is reversed by applying $V_{WRITE}(0)$ on BL(BLB). In the array, BL and BLB are routed along the column while WWL runs along the row, which enables simultaneous programming of $+1$ and -1 in multiple cells in a row over a single cycle. Since the write path comprises 2 MTJs and an access transistor (as opposed to 1MTJ and 1 transistor in standard STT-MRAM), the write performance is lower. For instance, with V_{WRITE}=$1.45V$, we need ∼100ns write pulse to achieve successful programming, which is 1.47x higher than standard STT MRAM (∼68ns). Also bitcell write energy increases by 1.39x. However, in typical weight-stationary BNNs, the write operation is much less frequent compared to CiM, hence, write speed is traded-off to achieve higher CiM robustness (similar to previous designs [30])

2) Read Operation: Fig. 2 illustrates the read operation of CREST bit-cell. Let us first consider that MTJ_L and MTJ_R store AP and P, which corresponds to high resistance (R_{HIGH}) and low resistance (R_{LOW}), respectively. BL and BLB are driven to V_{READ} and WL is asserted (V_{WL}=V_{DD}=$1.2V$), while keeping WWL at 0. Since the bitlines are initially at 0, node1 and node2 are discharged to begin with (i.e. initial V_{n1} = V_{n2} = 0). On the application of V_{READ}, the bit-lines start charging node1 and node2 via their respective MTJs. With MTJ_L in the AP state (R_{HIGH}) and MTJ_R in the P state (R_{LOW}), the charging of node2 is faster than node1. With a proper choice

Input Encoding (IE)

V_{WL}	In'
0	0
V_{DD}	+1

Weight Encoding (WE)

MTJ_L	MTJ_R	W
AP	P	-1
P	AP	+1

Output Encoding (OE)

$I_{OUT} = I_{BLB} - I_{BL}$	O'
$-I_{HIGH}$	-1
$+I_{HIGH}$	+1

XNOR Truth Table

In	In'=(In+1)/2	W	O'	O=2O'–W
-1	0	-1	0	+1
-1	0	+1	0	-1
+1	+1	-1	-1	-1
+1	+1	+1	+1	+1

Fig. 4. Input, Weight, and Output Encoding schemes, and the resulting XNOR truth table.

of V_{READ}, M1 turns on first, leading to high current in the left (AP) branch. Also, with M1 (and M3) ON, V_{n1} is pulled down closer to 0, which prevents M2 from turning ON. This yields a very low current in the right (P) branch. As M2 goes to OFF state, V_{n2} reaches almost V_{READ} in the steady state, sustaining the ON state of M1 and high current in the AP branch, which can be sensed. Similarly, if MTJ_L and MTJ_R store P and AP, respectively, very low current appears on the left branch and high readable current on the right one (Fig. 2). In summary, the cross-coupling action yields a considerable improvement in the ratio of high and low currents (I_H/I_L) due to I_H flowing in the AP branch, and I_L (primarily driven by an OFF transistor) flowing in the P branch. These currents are sensed at BL and BLB and then passed through a current-subtractor to achieve output current, $I_{OUT} = I_{BLB} - I_{BL}$. Details of the current subtractor design can be found in [30]. Essentially, I_{OUT} is positive for P-AP and negative for AP-P state.

There are two important points to note here. First, I_H and I_L flow through AP and P branches respectively, as opposed to standard MRAM. However, the distinguishability is enhanced significantly. Second, MTJs are influenced with current-driven STT during read, which has the potential to accidentally disturb the stored MTJ state. As the read current direction in our bit-cell is anti-parallelizing and cross-coupling ensures considerable reduction in P branch current, disturb current is reduced, leading to RDM boost (details later) and allows for an increased V_{READ} range to operate in, as opposed to standard MRAMs.

Benefits of the cross-coupled action are possible only with appropriate V_{READ} such that the access transistor in the AP branch (M1 in Fig. 2) can be turned ON. To understand this, we show the currents in the P and AP branches as a function of V_{READ} (Fig. 3). When $V_{READ} < V_{th}$, both M1-M2 are OFF, and nA currents flow through both branches. As $V_{READ} \geq V_{th}$, both transistors start to turn on and the cross-coupling action begins to have a prominent effect. Further V_{READ} increase makes cross-coupling stronger, hence a window opens up between I_H and I_L with orders of magnitude difference between them. A safe operating voltage range for read is marked with a dark rectangle in Fig 3. We choose V_{READ}=0.66V for our simulations where I_H/I_L is maximum (8150).Even with deviation from the target V_{READ} due to non-ideal IR drops, the ratio is still a few thousands, better than standard MRAMs.

B. CREST-CiM Array

To achieve CiM of MVM of the signed binary inputs and weights, we utilize the weight, input, and output encoding

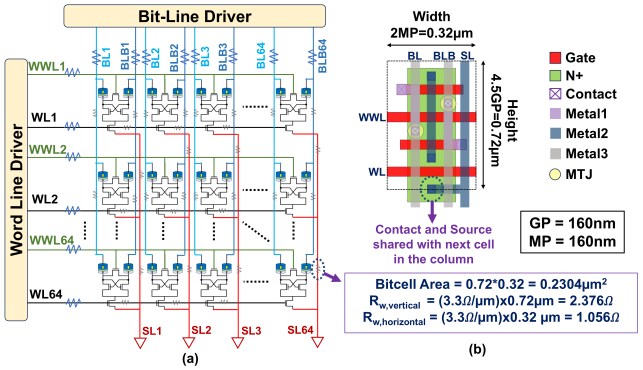


Fig. 5. CREST-CiM Crossbar Array and bitcell layout. (a) 64x64 crossbar array with non-idealities. WL,WWL are routed horizontally and BL,BLB,SL vertically. (b) bitcell layout, area and distributed wire resistance calculation based on resistance per unit length for 45nm technology node [26]

shown in Fig. 4. We transform the inputs $In \in \{-1, +1\}$ to $In' \in \{0, 1\}$ domain following the NANDNet architecture i.e. In'=$0.5(In + 1)$. Note, however, unlike the NANDNet design, weights are stored in the $\{-1, +1\}$ regime. We apply In' on WL, such that In'=+1 corresponds to WL voltage (V_{WL} =V_{DD}=$1.2V$), while In'=0 implies V_{WL}=0. Multiple inputs are applied simultaneously and the bit-cell currents naturally add up on BL and BLB. The BL and BLB currents are subtracted using a subtractor (discussed in the previous sub-section), which yields $O' = \sum_{k=1}^{n} In'.W$ (where n is the number of inputs applied). O' is digitized using analog-to-digital converters (ADCs). To obtain the final output O from digitized O', the following simple transformation can be derived.

$$\sum_{k=1}^{n} In.W = 2\sum_{k=1}^{n} In'.W - \sum_{k=1}^{n} W \quad (1)$$

$$\implies O = 2O' - \sum_{k=1}^{n} W \quad (2)$$

The multiplication of digital O' by 2 involves a simple left-shift operation. The sum of weights in the column can be pre-computed before deploying the weights in the arrays, and can be molded into the layer biases, averting any overhead. Note, this approach averts the dynamic computation of the sum of inputs (which needs an adder tree), as in the NANDNet architecture [7].

Let us consider an XNOR operation example, where 8 inputs are asserted in a single cycle (PWA of 8) to an ideal 64×64 crossbar array. Assume all the 8 inputs are +1 and all 8 corresponding bit-cells in a column have +1 weights. This yields $I_{BLB} = +8I_{HIGH}$,I_{BL}=0 $\implies$ $I_{OUT} = 8I_{HIGH}$. If all 8 bitcells have -1 weights, $I_{BLB} = 0, I_{BL} = 8I_{HIGH} \implies I_{OUT} = -8I_{HIGH}$. Consider another example when the activation pattern is In=$[-1, +1, +1, -1, +1, -1, +1, +1]$ $\implies$ In'= $[0, 1, 1, 0, 1, 0, 1, 1]$ and the 8 bitcells have weights, W=$[-1, +1, -1, -1, +1, -1, +1, +1]$. This leads to $I_{BLB} = 4I_{HIGH}$ and $I_{BL} = I_{HIGH}$ resulting in $I_{OUT} = +3I_{HIGH} \implies O' = +3$ (the dot product of In' and W).

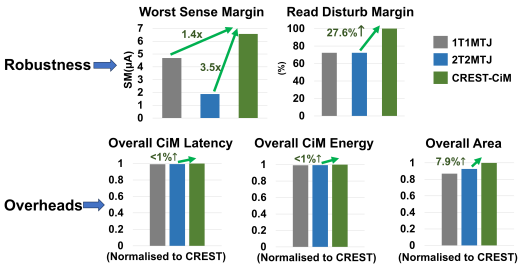


Fig. 6. Summary of CiM Robustness, and Latency-Energy-Area analysis

From (2) we get, $O = 2 \times (+3) - 0 = 6$, which is the XNOR output for this input-weight combination. Generally, if m and n bitcells are contributing to BLB and BL currents respectively, $I_{OUT} = (m-n)I_{HIGH}$, where $O' = m - n$.

IV. Results

We evaluate non-ideal 64×64 crossbar arrays with process variations and driver/wire resistances at WL,BL,BLB (Fig. 5a) to obtain the robustness and energy/latency/area of CREST-CiM. SL in these arrays are biased at virtual ground using an OP-AMP, following the design in [31]. MgO thickness, t_{MgO}=1.3nm is chosen to reduce MTJ current. The simulation parameters are summarized in Table I. We follow PWA of 8 to minimize non-ideal effects and lower ADC cost by using 3-bit Flash ADCs. All 64 columns of the asserted rows are activated to maximize column-parallelism. Currents are sensed from BL and BLB, and then passed through a current subtractor and a comparator, as used in [30] to obtain I_{OUT}, which then passes through ADC for extracting the digital output. We also present comparisons against 2 existing MTJ-based CiM designs: a 1T-1MTJ bit-cell with dummy columns and a 2T-2MTJ Differential bit-cell without any cross-coupling. Note that dummy columns are employed to mitigate the influence of I_L (as demonstrated in [32]), resulting in $I_{OUT} = I_{SL} - I_{SL,dummy}$, where the dummy currents are subtracted from each column's currents in the 1T-1MTJ baseline. For 2T-2MTJ baseline, $I_{OUT} = I_{BL} - I_{BLB}$, similar to CREST bit-cell but I_{BL} and I_{BLB} interchanged. For a fair comparison, we use the same parameters across all designs and optimize V_{READ} for baselines to match their I_H to the CREST bit-cell $I_H = 21.3\mu A$. A summary of the comparison is shown in Fig. 6.

A. Layout and Bitcell Area

Fig. 5b shows the CREST bit-cell layout based on 45nm technology design rules and predictive models [28], [29]. Transistors are optimized for drain-and-source sharing, with WL and WWL routed horizontally and BL, BLB, and SL routed vertically.

B. Sense Margin (SM) Analysis

PWA8 is applied to 8000 different input-weight combinations on the crossbar array and I_{OUT} is extracted for each column to calculate SM using the following equation,

$$SM = \frac{I_{OUT,a|min} - I_{OUT,a-1|max}}{2} \quad (3)$$

where, $I_{OUT,a|min}$ and $I_{OUT,a-1|max}$ are the minimum and maximum currents corresponding to one output and its preceding output, respectively. This quantifies distinguishability between two neighboring output states. Different input-weight combinations may correspond to the same output, yielding different output currents due to non-idealities. Therefore a single output state is mapped to a range of output currents instead of a single ideal output current, reducing SM from ideal case. Also, non-idealities cause unwanted voltage drops along BL and BLB based on their input-weight-dependence. We estimate the minimum possible effective V_{READ} on a bitcell, using the following equation:

$$V_{min,eff} = V_{READ} - 8I_H(R_D + 55R_w) - I_H R_w \sum_{k=1}^{8} k \quad (4)$$

where, R_D is the driver resistance,R_w is the distributed wire resistance. The extreme case is considered in (4) that all 8 bit-cells draw I_{HIGH} through either BL or BLB (worst case) so that the lower bound on effective V_{READ} is obtained. Our choice of V_{READ}=0.66V with $I_H = 21.3\mu A$ and $I_L = 2.61nA$, yields a minimum effective $V_{READ} = 0.593V$ which is in the safe region (see Fig. 3) with $I_H/I_L = 7400$. Due to large I_H/I_L ratio of CREST bit-cell, best- and worst-case non-ideal SM of $9.38\mu A$ and $6.56\mu A$ is achieved respectively (Fig. 7) compared to $4.68\mu A$ and $1.87\mu A$ for 1T-1MTJ and 2T-2MTJ cells respectively, as their $I_L = I_{AP} = 3.87\mu A$, orders higher than CREST-CiM. Higher SM implies better resilience of CREST-CiM against circuit non-idealities due to low current being dragged down to a few nA, leading to low IR drops. Also, $I_{OUT} = I_H - I_L$ almost equalling I_H in CREST bitcell causes less degradation from ideal current.

C. Read Disturb Margin (RDM) Analysis

It is imperative to ensure accidental read disturb does not occur and weights are retained. For assessing this, we use RDM, which is given by, $RDM = (I_{CR} - I_{MTJ})/I_{CR} \times 100\%$. Here, I_{CR} and I_{MTJ} denote critical switching current and actual MTJ current respectively. Operating closer to I_{CR} increases the probability of read disturb. Considering the read current direction in all 3 bit-cells are anti-parallelizing, we use $I_{CR} = 75.96\mu A$ for P→AP switching and calculate RDM as 72.35% for the two baselines, while for CREST bit-cell it is 99.996%, a 27.6% improvement. Cross-coupling action reduces the anti-parallelizing current in P branch down to $2.61nA$ for CREST, while for baselines, $I_P = 21.3\mu A$. This results in RDM boost for CREST.

D. Energy-Latency-Area Analysis

We analyze energy and area overheads of CREST-CiM considering peripheral circuits (row decoders, analog subtractors, 3-bit Flash ADCs per column, partial sum accumulators). We perform bitcell level latency analysis to approximate crossbar array performance. The CREST bitcell area is calculated as $0.2304\mu m^2$, which is 1.5x(3x) times that of 2T2MTJ(1T1MTJ)

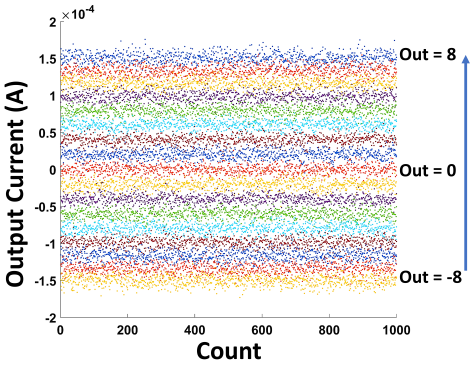


Fig. 7. Monte Carlo simulation results (1000 runs) for CREST-CiM. Low spread around most predominant output=0 can be observed.

bitcell respectively. However, at the array level with peripherals, CREST-CiM only comes with $7.9\%(14.9\%)$ overall area overhead over 2T2MTJ(1T1MTJ), primarily due to ADCs dominating overall-area. Also, CREST-CiM incurs just $< 1\%$ CiM latency and CiM energy overheads on both baselines.

E. Process-Induced Variations

Through 1000 Monte Carlo (MC) simulations per output state for CREST-CiM and 2T-2MTJ baseline, we perform variation analysis considering- (i)$25mV$ standard deviation (σ) of transistor threshold voltage, V_{th}, (ii) 1.5% σ of MTJ oxide thickness, t_{MgO}, and (iii) 5% of minimum metal width (65nm for 45nm technology node) as the σ of MTJ diameter [13]. Simulation results for CREST-CiM are illustrated in Fig. 7, showing minimal overlap between neighboring states, especially for lower outputs like the most predominant output 0. This is because of the increased SM of CREST-CiM. Overlaps are higher in 2T-2MTJ baseline design, hence, probability of errors under variation is higher in 2T-2MTJ compared to CREST-CiM, which translates to higher inference accuracy with our design as we will show in the next section.

V. System-Level Accuracy Analysis

In this section, we analyze the impact of circuit non-idealities and process-induced variations across the 3 mentioned designs with inference accuracies of trained ResNet-18 BNN on CIFAR-10 dataset.

A. Evaluation Framework

We design a customized crossbar simulator that exactly and self-consistently solves Kirchhoff's current and voltage laws in tandem with the bitcell I-V characteristics to compute the crossbar array output currents. The framework accounts for non-idealities such as wire resistance (R_{wire}), driver resistance (R_{driver}), sink resistance (R_{sink}), and device non-linearities. Using SPICE simulations, we derive the individual bitcell currents for the three bitcells analyzed in this work as functions of their terminal voltages, which are then organized into look-up tables (LUTs). These LUTs are integrated with the customized Xbar simulator to compute the non-ideal output currents. The simulator is implemented in PyTorch, ensuring seamless and

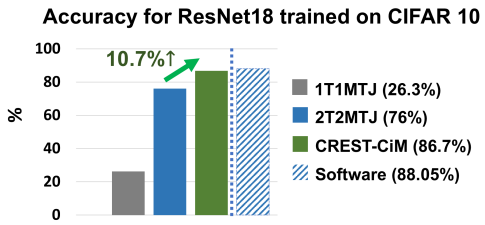


Fig. 8. Inference Accuracy with ResNet18 trained on CIFAR-10 dataset. CREST-CiM achieves only 1.35% degradation from software accuracy.

efficient integration into DNN workflows. Validation against SPICE simulations confirms the simulator's accuracy, with a maximum error of $< 2\%$.

To model variations in individual bitcell currents, we employ a Gaussian distribution. Using the findings from Section IV-E, we determine the standard deviation (σ) of the distribution and incorporate it into the bitcell currents as follows to account for variations: $I_{with-variations} = I_{nominal} * \mathcal{N}(1, \sigma)$. The derived σ values are approximately 17% and 12% for the two baseline cells and the CREST-CiM bitcell, respectively. Thus the inference accuracy presented accounts for the crossbar non-idealities, device non-linearities and variations.

B. Accuracy Analysis

Following our evaluation method, inference with trained ResNet18 on CIFAR-10 dataset using the 3 designs shows considerable inference accuracy with CREST-CiM (86.7%) (compared to $26.3\%(76\%)$ for 1T-1MTJ(2T-2MTJ) baselines) due to its improved SM and variation resilience (Fig. 8). This is significant considering only 1.35% degradation from software accuracy (88.05%).

VI. Conclusion

We propose CREST-CiM which addresses the low distinguishability problem in STT-MRAMs via cross-coupling of complementary MTJs. CREST-CiM significantly reduces the current associated with the 0 state, enhancing I_H-I_L ratio in bitcell, reduces the effect of hardware non-idealities and achieves a large boost in the sense margin. We also show improved process-variation resilience and enhancement in read disturb margin of the CREST-CiM design. The improvements in computational robustness achieved by CREST-CiM translate to near-software inference accuracy (86.7%) for ResNet18 BNN on CIFAR-10 dataset.

Acknowledgement

This work was funded in parts by CoCoSys, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA; by ME Commons through the SCMC Hub as a part of CHEETA, and by Raytheon.

REFERENCES

[1] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09, pp. 13 693–13 696, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/7123

[2] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and trends of sram-based computing-in-memory for ai edge devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1773–1786, 2021.

[3] X. Sun, S. Yin, X. Peng, R. Liu, J.-s. Seo, and S. Yu, "Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1423–1428.

[4] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11623

[5] A. Ding, Y. Qiao, and N. Bagherzadeh, "Bnn an ideal architecture for acceleration with resistive in memory computation," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 2, pp. 281–291, 2023.

[6] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[7] H. Kim, J. Sim, Y. Choi, and L.-S. Kim, "Nand-net: Minimizing computational complexity of in-memory processing for binary neural networks," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 661–673.

[8] X. Sun, R. Liu, X. Peng, and S. Yu, "Computing-in-memory with sram and rram for binary neural networks," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2018, pp. 1–4.

[9] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, 2020.

[10] G. Burr, R. Shelby, C. di Nolfo, J. Jang, R. Shenoy, P. Narayanan, K. Virwani, E. Giacometti, B. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element," in *2014 IEEE International Electron Devices Meeting*, 2014, pp. 29.5.1–29.5.4.

[11] S. Jung, H. Lee, S. Myung, H. Kim, S. K. Yoon, S.-W. Kwon, Y. Ju, M. Kim, W. Yi, S. Han, B. Kwon, B. Y. Seo, K. Lee, G. Koh, K. Lee, Y. Song, C. Choi, D.-H. Ham, and S. J. Kim, "A crossbar array of magnetoresistive memory devices for in-memory computing," *Nature*, vol. 601, pp. 211 – 216, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:245883891

[12] L. Chang, X. Ma, Z. Wang, Y. Zhang, Y. Xie, and W. Zhao, "Pxnor-bnn: In/with spin-orbit torque mram preset-xnor operation-based binary neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2668–2679, 2019.

[13] K. Cho, A. Malhotra, and S. K. Gupta, "Xnor-vsh: A valley-spin hall effect-based compact and energy-efficient synaptic crossbar array for binary neural networks," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 9, no. 2, pp. 99–107, 2023.

[14] S. Angizi, Z. He, A. Awad, and D. Fan, "Mrima: An mram-based in-memory accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 1123–1136, 2020.

[15] S. Ikeda, J. Hayakawa, Y. Ashizawa, Y. M. Lee, K. Miura, H. Hasegawa, M. Tsunoda, F. Matsukura, and H. Ohno, "Tunnel magnetoresistance of 604% at 300k by suppression of ta diffusion in cofebmgocofeb pseudo-spin-valves annealed at high temperature," *Applied Physics Letters*, vol. 93, no. 8, p. 082508, 08 2008. [Online]. Available: https://doi.org/10.1063/1.2976435

[16] I. Chakraborty, M. Fayez Ali, D. Eun Kim, A. Ankit, and K. Roy, "Geniex: A generalized approach to emulating non-ideality in memristive xbars using neural networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[17] T. Sharma, C. Wang, A. Agrawal, and K. Roy, "Enabling robust sot-mtj crossbars for machine learning using sparsity-aware device-circuit co-design," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2021, pp. 1–6.

[18] T.-N. Pham, Q.-K. Trinh, I.-J. Chang, and M. Alioto, "Stt-bnn: A novel stt-mram in-memory computing macro for binary neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 12, no. 2, pp. 569–579, 2022.

[19] H. Wang, W. Kang, B. Pan, H. Zhang, E. Deng, and W. Zhao, "Spintronic computing-in-memory architecture based on voltage-controlled spin–orbit torque devices for binary neural networks," *IEEE Transactions on Electron Devices*, vol. 68, no. 10, pp. 4944–4950, 2021.

[20] C. Wang, J. Victor, and S. K. Gupta, "Comparative evaluation of memory technologies for synaptic crossbar arrays – part i: Robustness-driven device-circuit co-design and system implications," 2024. [Online]. Available: https://arxiv.org/abs/2307.04261

[21] A. Aziz, N. Shukla, S. Datta, and S. K. Gupta, "Coast: Correlated material assisted stt mrams for optimized read operation," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 1–6.

[22] T.-X. Shi, B. Liu, and H. Cai, "Differential mram sensing scheme with magnetoresistance boosted 4t-2mtj bit-cell," *IEEE Transactions on Magnetics*, vol. 59, no. 11, pp. 1–5, 2023.

[23] H. Cai, Y. Guo, B. Liu, M. Zhou, J. Chen, X. Liu, and J. Yang, "Proposal of analog in-memory computing with magnified tunnel magnetoresistance ratio and universal stt-mram cell," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 4, pp. 1519–1531, 2022.

[24] J. Song, H. Dixit, B. Behin-Aein, C. H. Kim, and W. Taylor, "Impact of process variability on write error rate and read disturbance in stt-mram devices," *IEEE Transactions on Magnetics*, vol. 56, no. 12, pp. 1–11, 2020.

[25] D. Worledge, G. Hu, D. W. Abraham, P. Trouilloud, J. Nowak, S. Brown, M. Gaidis, and R. Robertazzi, "Spin torque switching of perpendicular ta cofeb mgo-based magnetic tunnel junctions," *Applied physics letters*, vol. 98, no. 2, 2011.

[26] P. Moon, V. Chikarmane, K. Fischer, R. Grover, T. A. Ibrahim, D. Ingerly, K. J. Lee, C. Litteken, T. Mule, and S. Williams, "Process and electrical results for the on-die interconnect stack for intel's 45nm process generation." *Intel Technology Journal*, vol. 12, no. 2, 2008.

[27] X. Fong, S. K. Gupta, N. N. Mojumder, S. H. Choday, C. Augustine, and K. Roy, "Knack: A hybrid spin-charge mixed-mode simulator for evaluating different genres of spin-transfer torque mram bit-cells," in *2011 International Conference on Simulation of Semiconductor Processes and Devices*, 2011, pp. 51–54.

[28] K. Mistry, C. Allen, C. Auth, B. Beattie, D. Bergstrom, M. Bost, M. Brazier, M. Buehler, A. Cappellani, R. Chau, C.-H. Choi, G. Ding, K. Fischer, T. Ghani, R. Grover, W. Han, D. Hanken, M. Hattendorf, J. He, J. Hicks, R. Huessner, D. Ingerly, P. Jain, R. James, L. Jong, S. Joshi, C. Kenyon, K. Kuhn, K. Lee, H. Liu, J. Maiz, B. McIntyre, P. Moon, J. Neirynck, S. Pae, C. Parker, D. Parsons, C. Prasad, L. Pipes, M. Prince, P. Ranade, T. Reynolds, J. Sandford, L. Shifren, J. Sebastian, J. Seiple, D. Simon, S. Sivakumar, P. Smith, C. Thomas, T. Troeger, P. Vandervoorn, S. Williams, and K. Zawadzki, "A 45nm logic technology with high-k+metal gate transistors, strained silicon, 9 cu interconnect layers, 193nm dry patterning, and 100

[29] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "Freepdk: An open-source variation-aware design kit," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, 2007, pp. 173–174.

[30] N. Thakuria, A. Malhotra, S. K. Thirumala, R. Elangovan, A. Raghunathan, and S. K. Gupta, "Site cim: Signed ternary computing-in-memory for ultra-low precision deep neural networks," 2024. [Online]. Available: https://arxiv.org/abs/2408.13617

[31] A. Jaiswal, I. Chakraborty, A. Agrawal, and K. Roy, "8t sram cell as a multibit dot-product engine for beyond von neumann computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2556–2567, 2019.

[32] S. Yu, P.-Y. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in *2015 IEEE International Electron Devices Meeting (IEDM)*, 2015, pp. 17.3.1–17.3.4.

Graph-Guided Transfer Learning to Boost the Efficiency of System-Level Optimization of Analog/Mixed-Signal Circuits

Jiaqi Wang, Georges G.E. Gielen
ESAT-MICAS, KU Leuven, 3001 Leuven, Belgium
{jiaqi.wang, georges.gielen}@kuleuven.be

***Abstract*—This paper introduces a novel graph-guided transfer learning approach to boost the efficiency of system-level optimization of analog/mixed-signal circuits. The system-level optimization is based on Reinforcement Learning (RL) in combination with Graph Attention Networks (GAT). The results surpass state-of-the-art in efficiency and optimality. The key innovation is a graph similarity detection method that leverages embedded design knowledge to identify electrical similarities and trade-offs, enhancing knowledge transferability, even between dissimilar circuit architectures. Applied to the case study of 4th-order continuous-time Delta-Sigma analog-to-digital converters, the graph-based transfer learning framework enhances the RL sampling efficiency, reducing the amount of simulations by up to 11x, and improves the optimization results by 12.4% compared to optimization from scratch. As the framework accelerates knowledge transfer across different architectures, it can boost the optimization efficiency and improve the performance towards a broad range of analog/mixed-signal systems.**

***Index Terms*—mixed-signal circuit design automation, system-level optimization, reinforcement learning, transfer learning**

I. Introduction

Analog/mixed-signal circuit design remains to hold paramount importance in modern electronic systems, where the coexistence of analog and digital components is essential when processing real-world signals. An essential block is the Analog-to-Digital converter (ADCs), which must accurately sample and digitize analog signals while minimizing all sources of error such as noise, distortion, and other non-idealities. The performance of ADCs is typically evaluated by key metrics such as the Signal-to-Noise-and-Distortion Ratio (SNDR), Signal-to-Noise Ratio (SNR), Bandwidth (BW), Total Harmonic Distortion (THD), and Spurious-Free Dynamic Range (SFDR). These metrics reflect the signal quality, the noise performance, and the frequency response, all of which are critical to high-fidelity signal conversion. Meanwhile, as the need for low energy is growing, minimizing the power consumption has become a primary design objective. Achieving optimal performance requires balancing the demanding metrics with power efficiency to meet the needs of modern high-performance, low-energy systems [1].

An hierarchical approach is often employed to manage the complexity of analog/mixed-signal circuit design by dividing circuits into smaller, functional sub-blocks. This method allows for a better organization and manageability [2]. For example, in ADC design, sub-blocks include amplifiers, sample-and-hold circuits, quantizers, integrators, digital interfaces, etc [3]. System-level optimization is then critical to balance conflicting

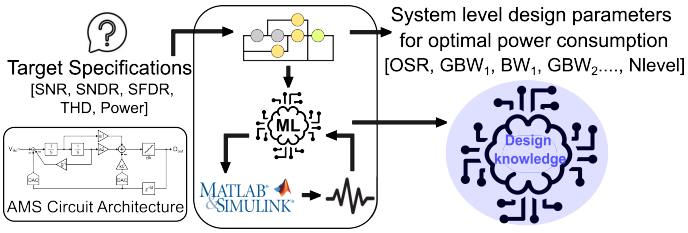


Fig. 1: Basic flow of AI-based system-level optimization of mixed-signal circuits.

design objectives such as performance, power consumption, and cost. It is challenging, though, due to the complexity of the hierarchical design space and the trade-offs involved [4].

Advanced optimization techniques are required to address such challenges effectively. Genetic algorithms (GAs) offer decent sampling efficiency but lack learning capabilities. Machine learning, and particularly reinforcement learning (RL), provides a promising alternative due to its adaptive learning and pattern recognition [5]–[7]. However, most current research on RL in circuit design focuses on device-level optimization, rather than on the entire system. The potential of RL for system-level optimization—where multiple interconnected sub-blocks influence the overall performance of the circuit—remains largely unexplored, particularly for mixed-signal circuits like ADCs. Our research addresses this gap by proposing an RL-based optimization framework for mixed-signal circuits (Figure 1), illustrated for the case of Delta-Sigma ADCs. Graph Attention Networks (GATs) are integrated to process the circuit topology, iteratively adjusting the design parameters to optimize the power while achieving the performance requirements. GATs enable the effective modeling of complex circuit relationships and facilitate knowledge transfer from previous tasks.

Additionally, we address a key limitation of existing research by investigating the transferability of knowledge learned by the RL agent. While prior work on transfer learning in analog circuit design has focused on evidently similar, smaller circuits at the device level, such as op-amps with varying numbers of stages [6], our approach also explores knowledge transfer across seemingly different and more complex circuit architectures, enabling their optimization more efficiently.

The contributions of this work are outlined as follows:

- Introduce the novel *application of RL with GAT* for the system-level optimization of mixed-signal circuits,

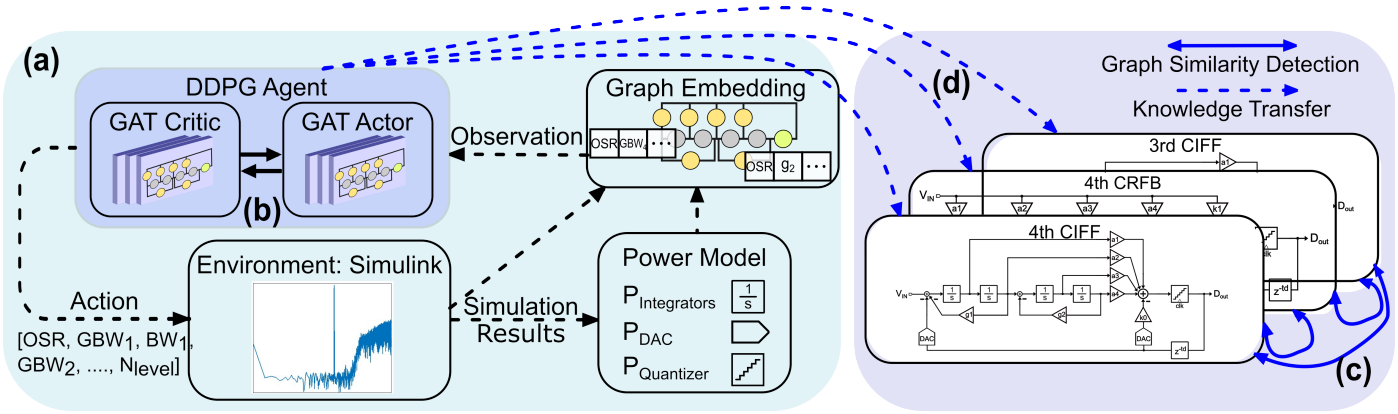


Fig. 2: Overview of the proposed framework for the system-level optimization of mixed-signal circuits. (a) The DDPG algorithm is implemented with (b) GATs as actor and critic to perform constrained optimization tasks. (c) The graph similarity detection method identifies the electrically similar architectures, which facilitates (d) the transfer of knowledge learned by the GAT agent (illustrated for $\Delta\Sigma$ architectures).

demonstrated for continuous-time Delta-Sigma Modulators ($\Delta\Sigma M$). Our approach achieves a reduction of up to 40% in power consumption, outperforming state-of-the-art methods by more effectively navigating the design space and capturing complex circuit dependencies.

- Propose a *novel graph similarity detection method* designed to identify analogous ADC architectures, surpassing the capabilities of non-experienced designers. Unlike previous research that focused on the knowledge transferability between evidently similar architectures [6], [8], [9], our approach leverages embedded design knowledge to detect electrical similarities and comparable design trade-offs in seemingly dissimilar architectures, enhancing the efficiency of knowledge transfer in RL optimization.
- Present a *transfer learning framework* that leverages graph information to transfer RL knowledge between circuit architectures. This framework drastically enhances the sample efficiency, reducing the number of required simulations by 1.7x-11x, and improving the optimization results by about 12.4%. This framework accelerates the optimization process and adapts to new design challenges, leading to better mixed-signal circuits.

II. Background

Previous work on knowledge transfer in analog circuit optimization has shown some potential but remains limited so far [6], [8]–[10]. GCN-RL investigated the knowledge transfer of RL with a graph convolutional network at the device level. However, its transferability was limited to circuits with the same topology at different technology nodes or to obviously similar architectures, such as between a 2-stage and a 3-stage transimpedance amplifier [6]. RL-based optimization was extended by transferring knowledge from schematic simulations to post-layout simulations and silicon-level measurements, thereby improving performance at various stages of the design flow [9]. KATO, using Bayesian optimization, also focused on transfers between closely related topologies [8]. While these papers demonstrate the feasibility of knowledge transfer in circuit design, their methods are constrained to topologies with clear architectural similarities. Recognizing these limitations in existing research, we extend the potential of knowledge transfer by proposing a novel approach. Using the Deep Deterministic Policy Gradient (DDPG) algorithm with Graph Attention Networks (GATs), our method optimizes mixed-signal circuits more efficiently and more effectively, overcoming the limitations of previous work. The next section will detail our methodology and its advantages.

III. Methodology

A. Problem Definition

The system-level design of mixed-signal circuits can be formulated as a constrained optimization problem to minimize the power consumption while meeting the system-level performance targets. The optimization problem is formulated as:

$$\min_{x\in\mathbb{R}^n} \quad Power(x) \tag{1}$$
$$\text{subject to} \quad s_i(x) \geq s_i^*, \ i = 1, 2, ..., M$$

where x represents the system-level design parameters in the mixed-signal circuits. s denotes the M key metrics of circuits while s^* represents the corresponding performance requirements. n denotes the number of system-level design variables.

B. Overview of the Proposed Framework

Figure 2 illustrates the proposed optimization framework, integrating four core elements for the efficient system-level optimization of hierarchical analog/mixed-signal circuits: (I) Reinforcement Learning (RL), (II) Graph Attention Networks (GAT), (III) graph similarity detection, and (IV) transfer learning. Each architecture is represented as a graph, converting circuit details into a structured format. During each RL step, **(I) the RL agent** receives observations and outputs actions corresponding to design parameters, followed by Simulink simulation to obtain the performance metrics. The state vector, encompassing the circuit information and simulation results, is embedded in the graph and fed into the RL agent. **(II) GATs**, acting as both actor and critic, process the graph information to guide the decision making, while rewards based on all metrics evaluate the design quality. Through iterative steps, the RL agent updates its policy in order to achieve designs

with minimal power consumption while meeting the required performance constraints. To enhance the sample efficiency, the framework employs **(III) graph similarity detection**. This process identifies electrical similarities between graph representations of different architectures, enabling the **(IV) transfer of knowledge** learned from optimizing one architecture to another architecture with similar design trade-offs. This transfer is facilitated by reusing parameters of graph neural networks, enhancing optimization efficiency across diverse circuit designs. In following sections, we elaborate on how these integrated components collectively optimize system-level designs, achieving significant advancements in performance and power efficiency across various circuit architectures.

C. Graph-Enhanced Reinforcement Learning Formulation

In this work, we employ the Deep Deterministic Policy Gradient (DDPG) algorithm as the RL agent for optimizing the mixed-signal circuits due to its relatively simple architecture and strong performance [11]. GATs is implemented as actor and critic to boost the performance of the optimization algorithm.

Action Space: Our action space comprises both global parameters and sub-block specifications in circuits. Sub-block specifications involve component-level performance metrics, while global parameters address system-wide considerations affecting overall circuit behavior and trade-offs. Table I details the description and range of each action parameter.

Reward Engineering: Our agent aims to minimize the power consumption while meeting the specified performance constraints s_i. The reward is computed as:

$$r_{constraints} = \sum_i \frac{s_i - s_i^*}{s_i + s_i^*}, where\ s_i\ is\ a\ hard\ constraint \tag{5}$$

$$R = \begin{cases} r_{constraints} - \beta * \frac{power - power^*}{power + power^*} & if\ r_{constraints} < 0 \\ 0.2 - 10 * \beta * \frac{power - power^*}{power + power^*} & else \end{cases} \tag{6}$$

where s_i^* represents the target for the corresponding specification. $power^*$ is a pre-defined reference value used to normalize the power consumption, ensuring that it falls within the same range [0,1] as the other normalized metrics. When the constraints aren't met, the reward combines the normalized specifications $r_{constraints}$ and the normalized power, weighted by the coefficient β (0.1), prioritizing achieving the performance targets above minimizing the power. If all targets are met, 0.2 is assigned instead of the normalized metrics, signifying no need for further optimization of the hard constraints. Additionally, the use of $10 * \beta$ emphasizes minimizing power consumption.

State Representations: Observations are crucial for the RL agent to understand the current state. We use graph neural networks to leverage circuits' architecture information, capturing both local and global details. Each sub-block in the circuit is represented as a node in the graph, with node features encoding its attributes. The node features are the state vector in the actor and the concatenation of the state and the action in the critic. The state vector for each sub-block comprises local and global states. Local states encompass key performance metrics specific to each component type. Global states include normalized simulation results and overall circuit performance indicators. Component types are also encoded to differentiate various elements within the circuit.

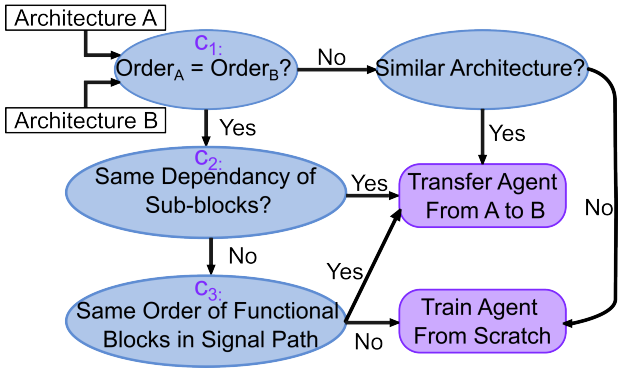


Fig. 3: Flowchart showing the similarity detection between circuit architecture A and architecture B.

Boosting RL with GATs: As shown in Figure 2, graph attention networks (GATs) serve as the actor and the critic in the DDPG algorithm. GATs leverage attention mechanisms to focus on relevant components and connections, effectively modeling dependencies between sub-blocks in the circuit for accurate and efficient system-level optimization [12]. We use 5 GAT layers to balance capturing local and global dependencies without overfitting, ensuring model transferability.

D. Graph-Guided System-Level Transfer Learning

The RL agent's learned knowledge from optimizing one mixed-signal circuit architecture can efficiently be transferred to optimize others. This paper introduces a new method to detect graph similarity between architectures, guiding an effective knowledge transfer, alongside a new critic architecture to enable knowledge transfer between circuits of different dimensions. It will be explained next.

1) Graph Similarity Detection: In our approach, circuits are represented as graphs, where edges depict signal paths between electrical components. The graph similarity between different mixed-signal circuits assesses the structural resemblance, indicative of comparable design trade-offs. Figure 3 illustrates the process of graph similarity detection. This process involves three key criteria, evaluated in sequence:

C_1 **- Circuit Order Comparison:** First, we compare the order of the two circuit architectures. As shown in the flowchart (Figure 3), if the orders are different, we assess whether the architectures are similar despite the order difference. Circuits of different orders typically have varying levels of complexity, and design knowledge may not directly translate between them due to differences in requirements, constraints, and optimization criteria. However, successful knowledge transfer can occur between circuits of different orders if they have a similar overall architecture. An example is the comparison between a 3rd-order CIFF $\Delta\Sigma M$ (Figure 5(a)) and a 4th-order CIFF $\Delta\Sigma M$ (Figure 5(b)) in section IV.

C_2 **- Sub-block Dependencies:** If the orders are the same, we next evaluate the sub-block dependencies, assessing whether the circuits have equivalent functional block relationships, as shown in Figure 3. This criterion examines the number and

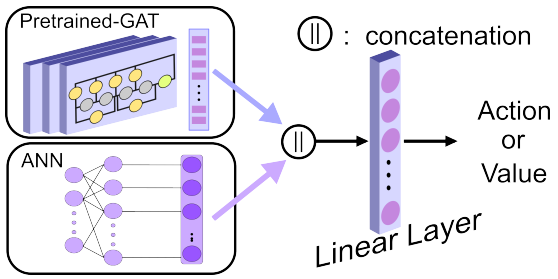


Fig. 4: The architecture of the critic during the transfer of the RL algorithm between circuits with a different number of system-level design variables.

arrangement of active elements (e.g., amplifiers, comparators) within signal paths. The interdependence of these elements influences the relationships between design variables and their impact on circuit performance.

C_3 **- Signal Path Analysis:** If the sub-block dependencies differ, we proceed to analyze the order of functional blocks in the signal path. This final criterion, as shown in Figure 3, examines the sequence of functional blocks within each signal path. Circuits with signal paths traversing through similar functional blocks in the same order are considered more alike. This similarity often results in comparable performance characteristics, affecting factors like frequency response, stability, and noise performance. Based on these criteria, the flowchart in Figure 3 determines whether to transfer the agent from Architecture A to B or to train a new agent from scratch.

This graph similarity detection flow will be illustrated with specific design cases in Section IV.B.(2).

2) Transfer Learning: If the circuit architectures are similar, the GAT actor and critic, trained for one circuit, can be applied to another circuit. The significance of the action in system-level circuits optimization lies in the limited number of states, with only a few specifications being critical. Thus, the entire action vector is concatenated with the state vector as node features at the input layer of the critic GAT. Consequently, node feature dimensions in the critic can vary with different circuits. To address this, an additional ANN (Figure 4) is used to extract features for the varying design parameters, maximizing the reuse of pre-trained layers. This approach enhances the system-level optimization performance, as demonstrated in the transfer from a 3rd-order to a 4th-order CIFF $\Delta\Sigma M$ in the next section.

IV. Experimental Results

The proposed optimization framework is demonstrated for the case study of continuous-time Delta-Sigma ($CT\Delta\Sigma$) ADCs, which offer high-resolution, low-power signal conversion for various applications [13]. Recent works have explored different approaches for $\Delta\Sigma M$ synthesis, including artificial neural networks requiring extensive datasets [14] and Bayesian optimization [15]. These methods, however, lack the ability to transfer knowledge between different architectures, a key advantage of our graph-guided reinforcement learning approach.

Design Parameters	Full Name	Range
OSR	Over sampling ratio	$[4, 40]$
k_i	Gain-Bandwidth coefficients of integrator i, $GBW_i = k_i * F_s$	$[0.5, 4]$
$Gain_i$	Gain of integrator i	$[25dB, 65dB]$
a_i	Loop filter coefficients	$[0.9 * a_i^*, 1.1 * a_i^*]$
g_i	Resonator coefficients	$[0.9 * g_i^*, 1.1 * g_i^*]$
N_{level}	Number of quantization level in quantizer	[1,31]

TABLE I: Summary of the design parameters for the system-level optimization of continuous-time Delta-Sigma modulators.

TABLE II: Simulation settings

Parameter Name	Parameter Value
Input Signal Bandwidth (BW)	200 [MHz]
Input Signal Amplitude	-2.5 [dBFs]
Sampling Frequency F_s	2*BW*OSR
DAC Clock Jitter	100 [fs]
Excess Loop Delay (ELD)	0.7 $[t_s]$
DAC Mismatch	1%
Data Weight Average (DWA)	$1st$ order

A. System-level Optimization

The system-level design of $CT\Delta\Sigma$ ADCs can be formulated as a constrained optimization problem to minimize the power consumption while meeting the key performance metrics, such as the Signal-to-Noise-and-Distortion Ratio (SNDR), Signal-to-Noise Ratio (SNR), Total Harmonic Distortion (THD), Maximum Stable Amplitude (MSA) and Spurious-Free Dynamic Range (SFDR). The requirements for specifications can be seen in Table III. The SNDR target is 80dB for all $\Delta\Sigma M$. The optimization results provide a comparison of the power consumption across different circuit architectures, which also can serve as a driver for topology synthesis. The action space includes both global parameters, such as the oversampling ratio, and sub-block specifications, such as integrator gain-bandwidth and amplifier gain. Table I provides a summary of these design parameters. The range for the loop filter coefficients a_i and the resonator coefficients g_i is based on the synthesized values a_i^* and g_i^* from the Delta-Sigma toolbox in MATLAB, which assumes that all integrators are ideal.

The Simulink model applied for system-level simulation of $CT\Delta\Sigma$ ADCs contains multiple non-idealities to ensure the design fidelity [17], as detailed in Table II. These non-idealities can degrade performance metrics like SNDR and SFDR, and therefore must be considered during the optimization process. By incorporating these non-linearities, our model ensures that the optimization process generates specifications that remain achievable at the device level, unlike idealized approaches that can produce unrealizable results. The signal bandwidth is set to 200 MHz for our high-speed applications.

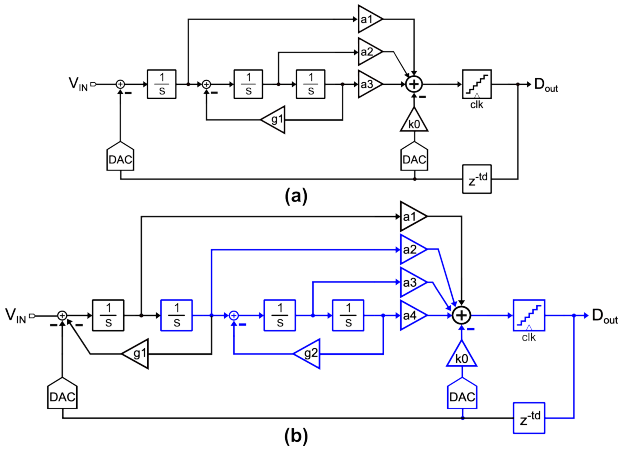


Fig. 5: Example data converters used in the experiments. (a) the 3rd-order CIFF (b) the 4th-order CIFF

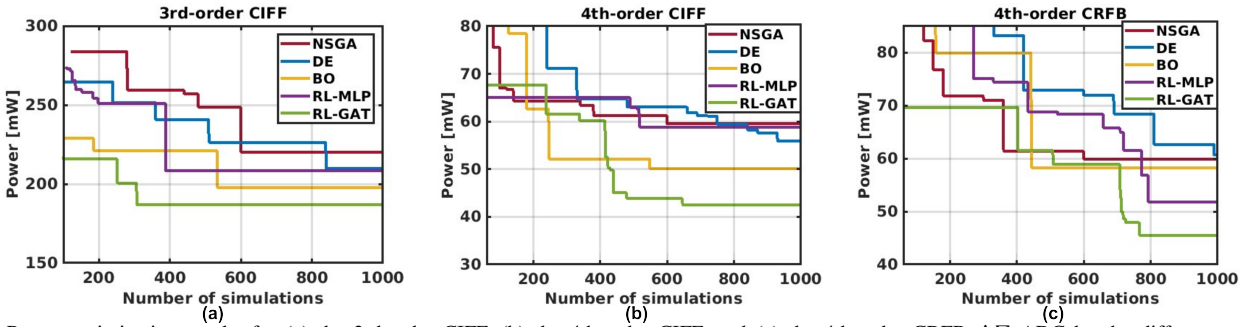


Fig. 6: Power optimization results for (a) the 3rd-order CIFF, (b) the 4th-order CIFF, and (c) the 4th-order CRFB $\Delta\Sigma$ ADC by the different optimization algorithms used (NSGA-II, DE, BO, RL-MLP and RL-GAT)

Optimization Algorithm	Data Converter Architecture	Performance Requirements	Optimized Power$[mW]$ and Improvement[%]	Data Converter Architecture	Performance Requirements	Optimized Power$[mW]$ and Improvement[%]	Data Converter Architecture	Performance Requirements	Optimized Power$[mW]$ and Improvement[%]
RL-GAT		SNDR≥80 $[dB]$	**186.71 (17.88%)**		SNDR≥80 $[dB]$	**42.43 (40.28%)**		SNDR≥80 $[dB]$	**45.52 (31.59%)**
RL-MLP		MSA≥2.5	208.17 (5.70%)		MSA≥2.5	58.76 (1.29%)		MSA≥2.5	51.79 (15.66%)
BO (Bayesian Optimization)	3rd-order	SNR≥82 $[dB]$	197.82 (11.26%)	4th-order	SNR≥82 $[dB]$	50.15 (18.68%)	4th-order	SNR≥82 $[dB]$	58.29 (2.76 %)
DE (Differential Evolution)	CIFF $\Delta\Sigma M$	SFDR≥85 $[dB]$	209.62 (5.09%)	CIFF $\Delta\Sigma M$	SFDR≥85 $[dB]$	55.96 (6.36%)	CRFB $\Delta\Sigma M$	SFDR≥85 $[dB]$	60.64 (-1.22%)
NSGA-II [16]		THD≤-82 $[dB]$	220.09 (0%)		THD≤-80 $[dB]$	59.52 (0%)		THD≤-80 $[dB]$	59.90 (0%)
MATLAB $\Delta\Sigma$ Toolbox			Fail (-)			Fail (-)			Fail (-)

TABLE III: Comparison of the results obtained for the system-level optimization of $\Delta\Sigma$ ADCs by different optimization methods (NSGA-II, DE, BO, RL-MLP, RL-GAT and the MATLAB $\Delta\Sigma$ toolbox).

To estimate consumption in state-of-the-art $\Delta\Sigma M$ and ensure device-level feasibility, we propose a power model based on ADC chips fabricated with advanced technologies [18]–[23]. Power consumption primarily comprises three parts: integrators, DACs, and quantizers. The power model is as follows:

$$P_{\int} = \sum_i C_{load} * \frac{GBW_i}{FoM} \tag{2}$$

$$P_{DAC} = p_{DAC} * N_{level} * f_{sample} * 10^{-15}\,[W] \tag{3}$$

$$P_{Quantizer} = p_{Quantizer} * N_{level} * f_{sample} * 10^{-15}\,[W] \tag{4}$$

Our designs use amplifiers with a high power efficiency as integrators with a FoM of 1GHz·pF/mW [18], [19]. Given the GHz-level sample frequencies, we adopt a representative load capacitance of 1pF for the integrators, within the typical range of 500fF-1.6pF. Based on contemporary $\Delta\Sigma$ ADC chips, where DAC power consumption typically ranges from 100-400fJ per conversion [22], [23] and quantizers consume 100-200fJ per conversion [20], [21], we adopt power models of 250fJ per conversion for DAC drivers (p_{DAC}) and 150fJ per conversion ($p_{Quantizer}$) for quantizers in our study.

The system-level optimization tasks were conducted on three example circuits: the 3rd-order CIFF $\Delta\Sigma M$ (Figure 5(a)), the 4th-order CIFF $\Delta\Sigma M$ (Figure 5(b)), and the 4th-order CRFB $\Delta\Sigma M$ (Figure 7(b)). The Simulink simulations for the 4th-order continuous-time $\Delta\Sigma M$ take approximately 30 seconds on a server with Intel® Xeon® Gold 6126 CPU. The runtime is attributed to the continuous operation requiring detailed analysis of analog components over continuous time intervals. NSGA-II [24], Differential Evolution (DE), a Gaussian Process-based Bayesian Optimization (BO) using Expected Improvement acquisition function, and RL with MLP (multilayer perceptron) implemented as actor and critic are selected as the reference optimization methods. The MLP has 5 layers to match the depth of the GAT architecture. Each algorithm runs for 1000 simulations, after which the optimization reaches saturation.

Figure 6 shows the optimization results for the three $CT\Delta\Sigma M$ architectures of Figure 5(a), 5(b) and 7(b). After a few simulation steps, RL-GAT consistently found solutions with lower power consumption than the reference optimization algorithms while meeting the performance requirement. RL-MLP, BO and DE achieves certain power reductions compared to NSGA-II, indicating some improvement. RL-GAT outperformed all, achieving power reductions of 17.88%, 40.28%, and 31.59%, excelling due to its ability to capture complex relationships within circuit graphs via attention mechanisms. This capability allows RL with GAT to navigate the design space more effectively, resulting in a significantly lower power consumption across all tested architectures.

The optimization results have also been compared with the widely used MATLAB Delta-Sigma toolbox, which is frequently employed by expert designers to synthesize $\Delta\Sigma$ ADCs [25]. The toolbox however has significant limitations as it doesn't account for real-world effects. In reality, the integrators have a finite gain and finite GBW as included in our model, leading to deviations in the noise transfer function (NTF). Therefore, the synthesized coefficients by the MATLAB toolbox cannot guarantee the optimized SNR, SNDR and SFDR. The deviation of the NTF due to the toolbox's simplifications is demonstrated by our experiment. Substituting the optimized loop filter coefficients a_i and the resonator coefficients g_i found by RL with the synthesized values (a_i^* and g_i^*) from the MATLAB $\Delta\Sigma$ toolbox resulted in a decreases in SNR, SNDR and SFDR by a few dB, failing to meet the performance requirement because of the non-idealities. Table III summarizes and compares the results achieved by RL-GAT, RL-MLP, BO, DE, NSGA-II, and the MATLAB $\Delta\Sigma$ toolbox.

B. Knowledge Transfer

1) Graph Similarity Detection: To demonstrate the effectiveness of our proposed graph similarity detection method, we applied it to three $CT\Delta\Sigma ADCs$ architectures: 4th-order CIFF, 4th-order CRFB, and 4th-order CIFF-B. Figure 7 illustrates

	The integrators in the signal paths of all order			
	1st-order	2nd-order	3rd-order	4th-order
$4th-order$ CIFF $\Delta\Sigma M$	1	1,2	1,2,3	1,2,3,4
$4th-order$ CRFB $\Delta\Sigma M$	4	3,4	2,3,4	1,2,3,4
$4th-order$ CIFF-B $\Delta\Sigma M$	4	3,4	1,3,4	1,2,3,4

TABLE IV: The sequence of integrators in each path for the three example circuits of Figure 7.

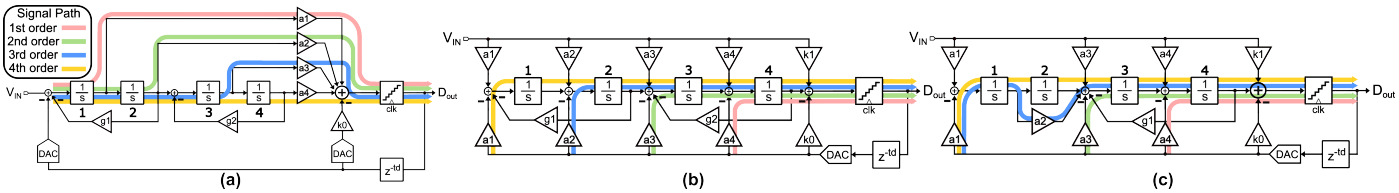


Fig. 7: The signal paths in (a) 4th-order CIFF $\Delta\Sigma M$, (b) 4th-order CRFB $\Delta\Sigma M$, (c) 4th-order CIFF-B $\Delta\Sigma M$

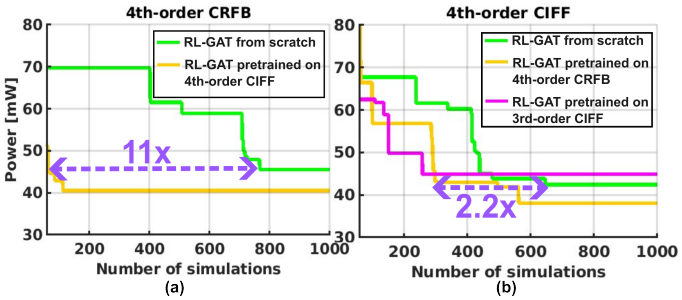


Fig. 8: Transfer learning results demonstrating the efficacy of RL-GAT in optimizing power consumption across different $\Delta\Sigma M$ architectures.

these architectures. Following the flowchart proposed in Figure 3, we first compared the order of the circuits. All three architectures are 4th-order, satisfying the first criterion for similarity. Next, we examined the dependency of amplifying devices. An nth-order $\Delta\Sigma M$ has n signal paths. The order of a signal path is determined by the number of integrators it contains. In the 4th-order CIFF and the 4th-order CRFB architectures, each amplifying stage operates independently within a single signal path. However, in the 4th-order CIFF-B architecture, amplifier $a1$ amplifies signals across both the 3rd- and 4th-order paths. This shared amplification causes the coupling of $a1$ and $a2$ in the CIFF-B architecture, affecting their interdependence. Given this difference in amplifier dependencies, we proceeded to the third criterion: the sequence of integrators in the signal path. As shown in Table IV, the 4th-order CIFF and the 4th-order CRFB architectures have signal paths traversing through the same integrators in the same order, indicating a high degree of similarity. Conversely, the 4th-order CIFF-B architecture differs significantly in its sequence of integrators compared to the other two. Our analysis identified strong similarities between the 4th-order CIFF and the 4th-order CRFB architectures. Additionally, we considered knowledge transfer between the same architecture of different orders (the 3rd-order CIFF to the 4th-order CIFF), as our flowchart allows for similarity detection between architectures of different orders if their overall structure is similar. To validate our method, we conducted knowledge transfer experiments between 4th-order CIFF and 4th-order CRFB architecture (bidirectionally), and from the 3rd-order to the 4th-order CIFF architecture. The following sections present results demonstrating the efficacy of our graph similarity detection in identifying knowledge transfer opportunities for $\Delta\Sigma M$ design optimization.

2) Transfer Learning: Figure 8(a) illustrates the optimization results for the 4th-order CRFB $\Delta\Sigma M$ using RL-GAT, leveraging the design knowledge learned from the 4th-order CIFF $\Delta\Sigma M$. The transferred agent can rapidly find good solutions with considerably lower power consumption, decreasing simulation steps by 11x and saving approximately 5.95 hours compared to the RL algorithm learning from scratch. The RL-GAT with knowledge also achieved optimal results, lowering the power consumption by $\sim$12.4% compared to the RL-GAT learned from scratch and $\sim$47.9% compared to NSGA-II. Similarly, knowledge transfer from the 4th-order CRFB $\Delta\Sigma M$ to the 4th-order CIFF $\Delta\Sigma M$ (Figure 8(b)) has reduced the number of simulation steps by 2.2x to achieve the optimal results, compared to RL-GAT learning from scratch. This improved sample efficiency confirms the highly comparable dependencies and trade-offs between the design variables in the 4th-order CRFB $\Delta\Sigma M$ and the 4th-order CIFF $\Delta\Sigma M$, which is also confirmed by the high graph similarity. The GAT effectively analyzes complex designs, identifying patterns within pre-learned circuits.

The transfer of knowledge from the 3rd-order CIFF $\Delta\Sigma M$ to the 4th-order CIFF $\Delta\Sigma M$ (Figure 8(b)) utilized the critic architecture presented in Section III.C.(2), leveraging identical circuit graphs (shown in Figure 5(a) and 5(b)) for direct deployment of the GAT knowledge. The simulation steps needed to achieve a good optimization result decrease by 1.7x compared to the RL algorithm learning from scratch, at the expense of a slight $\sim$5.6% increase in power consumption at the end of the optimization. This illustrated clearly the benefit of the transferred knowledge. Conversely, in identical transfer learning experiments, the RL-MLP demonstrated an ineffective knowledge transfer, failing to enhance power consumption or optimization speed and underscoring its limitations in applying learned design knowledge across architectures.

V. Conclusions

This paper has presented a novel graph-guided transfer learning approach for the system-level optimization of analog/mixed-signal circuits, demonstrated on continuous-time $\Delta\Sigma$ ADCs. The proposed method, combining Reinforcement Learning with Graph Attention Networks, has achieved up to 40% reduction in power consumption while meeting the performance requirements compared to traditional state-of-the-art optimization methods. The graph similarity detection method and transfer learning framework have been described to achieve knowledge transfer from one architecture to another, allowing to optimize faster while achieving substantial power savings without compromising the performance. This underscores the effectiveness of reinforcement learning and transfer learning in the system-level optimization of analog/mixed-signal circuits.

Acknowledgment

This work was funded by the European Research Council (ERC) under the Horizon 2020 program (grant n° 101019982 - AnalogCreate).

REFERENCES

[1] Dongyang Jiang, Sai-Weng Sin, Liang Qi, Guoxing Wang, and Rui P. Martins. Recent advances in high-resolution hybrid discrete-time noise-shaping adcs. *IEEE Open Journal of the Solid-State Circuits Society*, 1:129–139, 2021.

[2] Tom Eeckelaert, Raf Schoofs, Georges Gielen, Michiel Steyaert, and Willy Sansen. Hierarchical bottom-up analog optimization methodology validated by a delta–sigma a/d converter design for the 802.11a/b/g standard. In *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, page 25–30, 2006.

[3] Georges GE Gielen and Rob A Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1854, 2000.

[4] K. Francken, P. Vancorenland, and G. Gielen. Daisy: a simulation-based high-level synthesis tool for /spl delta//spl sigma/ modulators. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD - 2000. IEEE/ACM Digest of Technical Papers*, pages 188–192, 2000.

[5] Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. Learning to design circuits. *arXiv preprint arXiv:1812.02734*, 2018.

[6] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. Gcn-rl circuit designer: transferable transistor sizing with graph neural networks and reinforcement learning. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, DAC '20, 2020.

[7] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. Autockt: deep reinforcement learning of analog circuit designs. In *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, DATE '20, page 490–495, 2020.

[8] Wei W Xing, Weijian Fan, Zhuohua Liu, Yuan Yao, and Yuanqi Hu. Kato: Knowledge alignment and transfer for transistor sizing of different design and technology. *arXiv preprint arXiv:2404.14433*, 2024.

[9] Juzheng Liu, Shiyu Su, Meghna Madhusudan, Mohsen Hassanpourghadi, Samuel Saunders, Qiaochu Zhang, Rezwan Rasul, Yaguang Li, Jiang Hu, Arvind Kumar Sharma, Sachin S. Sapatnekar, Ramesh Harjani, Anthony Levi, Sandeep Gupta, and Mike Shuo-Wei Chen. From specification to silicon: Towards analog/mixed-signal design automation using surrogate nn models with transfer learning. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

[10] Yaguang Li, Yishuang Lin, Meghna Madhusudan, Arvind Sharma, Sachin Sapatnekar, Ramesh Harjani, and Jiang Hu. A circuit attention network-based actor-critic learning approach to robust analog transistor sizing. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2021.

[11] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[12] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[13] Johannes Wagner, Maurits Ortmanns, and José M. de la Rosa. Man or machine — design automation of delta-sigma modulators. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018.

[14] Pablo Díaz-Lobo, Gustavo Liñán-Cembrano, and José M de la Rosa. On the use of artificial neural networks for the automated high-level design of σ δ modulators. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[15] Jialin Lu, Yijie Li, Fan Yang, Li Shang, and Xuan Zeng. High-level topology synthesis method for δ-σ modulators via bi-level bayesian optimization. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[16] Manuel Velasco-Jiménez, Rafael Castro-López, and José M de la Rosa. High-level optimization of $\sigma\delta$ modulators using multi-objetive evolutionary algorithms. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1494–1497. IEEE, 2016.

[17] S Brigati et al. Modeling sigma-delta modulator non-idealities in simulink(r). In *1999 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 384–387 vol.2, 1999.

[18] Hui Liu, Xinpeng Xing, and Georges G. E. Gielen. An 85-mhz-bw asar-assisted ct 4-0 mash $\delta\sigma$ modulator with background half-range dithering-based dac calibration in 28-nm cmos. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(7):2405–2414, 2019.

[19] Chenming Zhang. *Broadband Continuous-time MASH Sigma-Delta ADCs*. Phd thesis 1 (tue), Electrical Engineering, September 2021. Proefschrift.

[20] Athanasios T. Ramkaj, Michiel S. J. Steyaert, and Filip Tavernier. A 13.5-gb/s 5-mv-sensitivity 26.8-ps-clk–out delay triple-latch feedforward dynamic comparator in 28-nm cmos. *IEEE Solid-State Circuits Letters*, 2(9):167–170, 2019.

[21] Julius Edler, Marcel Runge, Sebastian Linnhoff, and Friedel Gerfers. A 4.4 gs/s 220 mhz $\delta\sigma$ adc with a linearized back-gate controlled gmc filter. In *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 1–2, 2023.

[22] Qilong Liu et al. A 5gs/s 360mhz-bw 68db-dr continuous-time 1-1-1 filtering mash $\delta\sigma$ adc in 40nm cmos. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 414–416, 2022.

[23] Tao He, Michael Ashburn, Stacy Ho, Yi Zhang, and Gabor Temes. A 50mhz-bw continuous-time $\delta\sigma$ adc with dynamic error correction achieving 79.8db sndr and 95.2db sfdr. In *2018 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 230–232, 2018.

[24] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[25] Ovidiu Bajdechi, Georges E Gielen, and Johan H Huijsing. Systematic design exploration of delta-sigma adcs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(1):86–95, 2004.

ESM: A Framework for Building Effective Surrogate Models for Hardware-Aware Neural Architecture Search

Azaz-Ur-Rehman Nasir*, Samroz Ahmad Shoaib*, Muhammad Abdullah Hanif*, Muhammad Shafique
eBRAIN Lab, Division of Engineering, New York University Abu Dhabi, UAE
{an3525, ss14256, mh6117, muhammad.shafique}@nyu.edu

***Abstract*—Hardware-aware Neural Architecture Search (NAS) is one of the most promising techniques for designing efficient Deep Neural Networks (DNNs) for resource-constrained devices. Surrogate models play a crucial role in hardware-aware NAS as they enable efficient prediction of performance characteristics (e.g., inference latency and energy consumption) of different candidate models on the target hardware device. In this paper, we focus on building hardware-aware latency prediction models. We study different types of surrogate models and highlight their strengths and weaknesses. We perform a systematic analysis to understand the impact of different factors that can influence the prediction accuracy of these models, aiming to assess the importance of each stage involved in the model designing process and identify methods and policies necessary for designing/training an effective estimation model, specifically for GPU-powered devices. Based on the insights gained from the analysis, we present a holistic framework that enables reliable dataset generation and efficient model generation, considering the overall costs of different stages of the model generation pipeline.**

***Index Terms*—Neural Architecture Search, Surrogate Models, Hardware, Latency Prediction, Resource-Constrained Devices**

I. Introduction and Motivation

Deep Neural Networks (DNNs) are nowadays being used to serve a wide variety of tasks in different domains such as computer vision [1] and Natural Language Processing (NLP) [2] [3]. Moreover, the diverse set of devices used for DNN deployment in real-world application (e.g., embedded CPU and GPU devices, as well as specialized hardware accelerators [4]) brings a unique challenge, as DNNs result in different latency and energy numbers on different devices because of the differences in their hardware architecture and capabilities [5]. Neural Architecture Search (NAS), specifically, hardware-aware NAS (HW-NAS), techniques have been proposed to automatically search for DNNs based on the user-defined performance and hardware constraints [6]. A generic overview of hardware-aware NAS techniques is presented in Fig. 1.

Various HW-NAS techniques have been proposed in the literature. One prominent set of techniques focuses on simultaneous exploration and training of candidate models to converge to a suitable solution. In contrast, methods like Once-for-All (OFA) [5] adopt a decoupled strategy, where a supernet is first trained, followed by exploration of DNN configurations based on user-defined constraints, without involving additional training. The core benefit of the latter category lies in fast exploration, as it eliminates model training iterations. Building on this advantage, our work focuses on improving the exploration phase of OFA-based techniques to efficiently and accurately identify suitable models for deployment on target hardware, while satisfying all user-defined performance constraints.

Since deploying each candidate model on hardware to evaluate its performance is time-consuming, HW-NAS techniques

*These authors contributed equally to this work

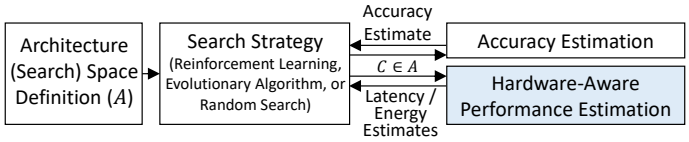


Fig. 1. A general overview of hardware-aware NAS which highlights the importance of hardware-aware performance (latency or energy) estimation for guiding the architecture search strategy.

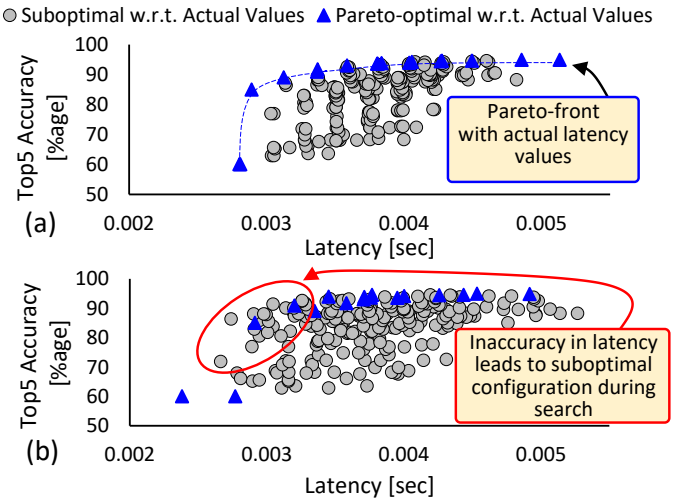


Fig. 2. Impact of inaccuracy in latency prediction on neural architecture search results. (a) illustrates the top-5 accuracy vs. latency plot of 243 different ResNet architectures generated by varying the depth values of the pre-trained ResNet50 supernet made available by Cai et al. [5]. The accuracy numbers are generated by evaluating each candidate model on the ImageNet validation set, and the latency numbers are generated by running the model on an RTX4090 GPU. (b) illustrates how the actual Pareto-optimal points can potentially move in the architecture space when the latency prediction is slightly inaccurate.

utilize surrogate models to efficiently estimate the performance or latency of DNNs on the target hardware [7] [8] [9] [10] [11]. Fig. 2 highlights the importance of using a precise latency prediction model to converge to models that lie on or near the actual Pareto front. Various techniques have been proposed for building surrogate models for HW-NAS. Earlier methods relied on proxy metrics (e.g., FLOPs) to estimate DNN performance [12] [13]. However, such metrics are hardware-agnostic and fail to accurately reflect a model's true performance or latency on the target hardware. To address this limitation, lookup table and ML-based techniques have been proposed for latency prediction. Lookup table-based techniques use an additive model where latency of each individual layer is taken from the lookup table (defined through profiling) and then the latencies of all the layers are accumulated to produce the latency of the complete model. These techniques result in suboptimal results as they lack in capturing the complex interactions between layers. The ML-based techniques encode the model architecture into an intermediate representation, which serves as input for performance predictors. The most common architecture encod-

ing types include one-hot [14] [15], and feature representation encodings [16] [17] [18]. These are used with models such as multilayer perceptron (MLP), linear regression, decision tree regression, and boosted decision tree regression [15] [18] [16] [17]. Other techniques leverage the graphical structure of DNNs to utilize Graph Convolutional Networks (GCNs) for building performance predictors [14] [19]. However, these ML models lack in providing accurate latency estimates as will be demonstrated in the following motivational case study.

A. Motivational Case Study

In this section, we highlight the impact of increasing the training set size on the accuracy of latency prediction models. For this analysis, we considered two different supernet architecture types: ResNet and DenseNet architectures. The configurations for both are presented in the experimental setup section (III.A). Note that the architecture space considered for ResNet supernet is much larger than that considered for DenseNet. For prediction model, we used MLP together with summary feature representation encoding from [11]. For each architecture type, we randomly sampled 24000 samples from the architecture space and executed them on an RTX4090 machine to gather architecture-latency pairs for predictor training and evaluation. We used 4000 samples for testing and 8000 or 20000 for training. Fig. 3(e) shows that, for both the architecture types, increasing the training set size from 8000 architecture samples to 20000 samples does not result in any significant improvement in the prediction accuracy. Moreover, as the architecture space considered for DenseNet supernet is much smaller than that of ResNet, the predictor results in significantly higher accuracy for DenseNet. We argue that the lower accuracy in the case of ResNet model is due to the overlapping representation of configurations in the representation space, and thus, *there is a need to explore alternative encoding schemes that can result in better accuracy for larger OFA architecture spaces.*

In addition to the above, we compare the time required to accurately compute the latency of a sample architecture on an RTX4090 GPU with the time needed to train a latency predictor. Fig.4(a) shows that the average latency measurement time per model is comparable to the time required to train a latency predictor with over 8,000 samples. Note that each latency measurement involves multiple inferences to account for fluctuations, as illustrated in Fig.4(b). *On one hand, this emphasizes the high cost of data acquisition, while on the other hand, it highlights the potential for an iterative methodology—comprising training, testing, and dataset extension if necessary—to ensure early exit and minimal overall model generation cost.*

B. Our Novel Contributions

To address the above challenges, in this paper, we make the following key contributions.

1) We propose ESM, a framework for building effective latency prediction models for HW-NAS algorithms. ESM targets layer/block-wise architecture spaces built on a fixed macro-architecture. The framework adopts a *train-evaluate-extend* approach, allowing it to efficiently converge on an accurate model that meets user-defined accuracy constraints while minimizing dataset collection costs.

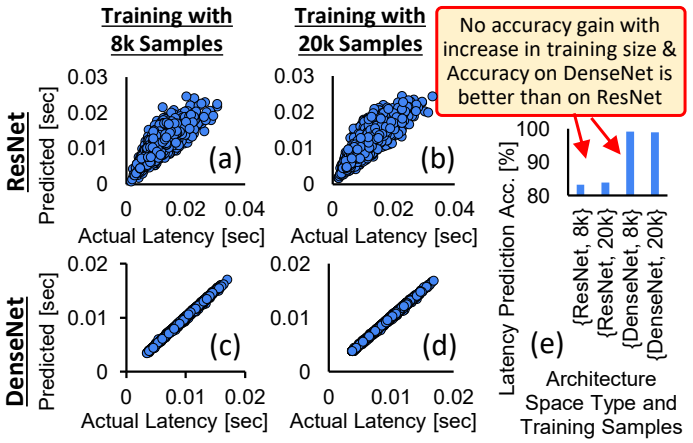


Fig. 3. Actual vs. predicted latency for 4k ResNet model configurations using (a) a predictor trained with 8,000 samples and (b) a predictor trained with 20k samples. (c) and (d) present the results for DenseNet configurations. (e) Comparison of the average accuracy of predictors trained with 8,000 samples vs. those trained with 20,000 samples for the two model types.

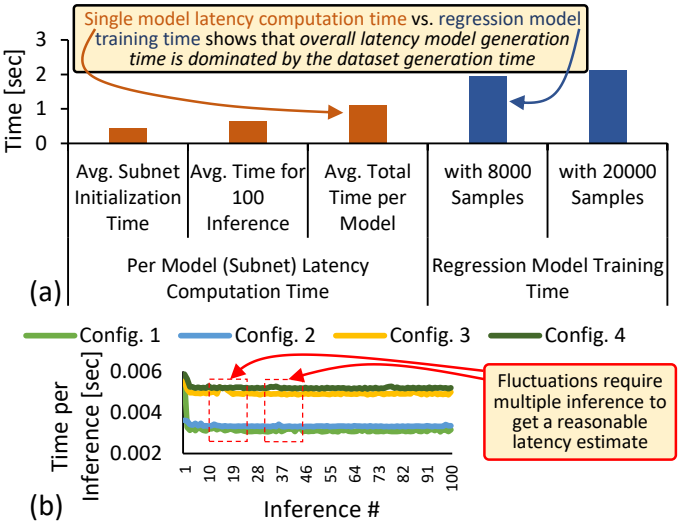


Fig. 4. (a) Comparison between the time required to accurately compute the latency of a model derived from the pre-trained ResNet50 supernet by Cai et al. [5] on an RTX4090 GPU and the training time for a latency predictor. On average, training a regression model of reasonable size on an RTX4090 GPU takes approximately the same amount of time as computing the latency of a model on the target hardware. (b) Latency values across inferences for different architectural configurations, illustrating the need for multiple iterations to obtain a reliable value.

2) We propose multiple different architecture encoding schemes specialized for building accurate latency prediction models for layer/block-wise operation space built on a fixed macro-architecture. Our encoding schemes provide a balance between accuracy and surrogate model complexity, without significantly increasing the need for larger training datasets to achieve high accuracy. This is achieved by enhancing the representation capability of the encoding while minimizing the risk of increased sparsity, which could otherwise require larger datasets.
3) Our results on a variety of models and devices demonstrate the effectiveness of our proposed encoding schemes over state-of-the-art encoding schemes, specifically for highly diverse architecture search spaces based on fixed macro-architectures.

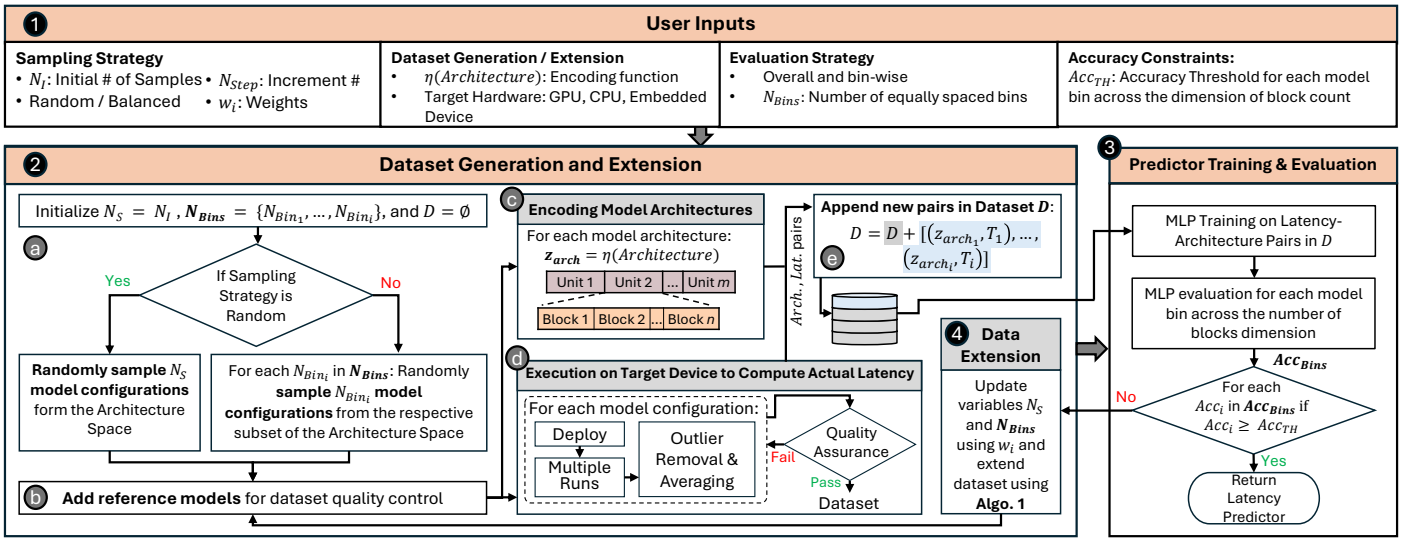


Fig. 5. Overview of our ESM Framework.

II. Methodology

A. Overview

Fig. 5 provides an overview of our novel framework that can produce an accurate latency predictor. The user defines number of initial samples, sampling strategy, encoding type, evaluation strategy, and accuracy constraints. Based on the selected sampling strategy, model configurations are sampled from the Architecture Space. The selected architectures are encoded, and also have their inference performed on the target hardware. These architecture-latency pairs make up the dataset. An MLP-based latency predictor is trained on this dataset, and the performance of the predictor is evaluated. If the user-defined accuracy constraints are met, the framework returns the latency predictor. Otherwise, the dataset is extended and the predictor is retrained until the accuracy constraints are met.

B. User Inputs

The user defines the following inputs for the framework (see ❶ in Fig. 5):

1) **Sampling strategy:** *random* or *balanced*.
2) **Target Hardware:** GPU, CPU or an embedded device.
3) $\mathbf{N_I}$: Number of initial samples to be sampled from the architecture space.
4) $\mathbf{N_{Step}}$: Number of additional samples that will be added to N_I during dataset extension.
5) $\mathbf{w_i}$: Weights assigned to below accuracy and above accuracy threshold bins.
6) η**(Architecture)**: Architecture encoding function
7) **Evaluation Strategy**: Overall or bin-wise accuracy
8) $\mathbf{N_{Bins}}$: Number of equally spaced bins along number of blocks (depth) dimension on which the dataset will be divided
9) $\mathbf{Acc_{TH}}$: Accuracy threshold for N_{Bins} to meet to pass the evaluation strategy

C. Dataset Generation

Once the user has defined their inputs, the framework proceeds based on the selected sampling (*random* or *balanced*) strategy (see ⓐ in Fig. 5). Moreover, based on the number of blocks in each model, the dataset is divided into equally spaced N_{Bins} bins.

1) **Random Sampling:** In this sampling strategy, we randomly select a given number of models from the entire architectural space. Fig 7 outlines the general structure of our models. Each model is made up of k units; a unit is composed of l number of blocks. For each block, user defines features that they want to vary. For example, in our experiments we fixed each ResNet model to have 4 units, and each unit could have 1 to 7 blocks. When randomly selecting for number of blocks of each unit, a Gaussian spread between the least and the most number of blocks is seen due to the central limit theorem. This bias in the sampling distribution forces the predictor to perform poorly for models present in the corner cases.
2) **Balanced Sampling:** To counter the bias that arises naturally with random sampling, the user can select *balanced* strategy. The entire architecture space is divided into equally spaced bins, N_{Bins}, based on the number of blocks. For each bin in N_{Bins}, equal number of models are sampled from the architecture space. This prevents the underrepresentation of the corner cases and allows the predictor to effectively learn their trends as well. Each block's kernel size and width expansion ratio is still randomly selected. Once models have been sampled from the architecture space, reference models (randomly selected from the architecture space) are added to the sample space. These reference models will be later used for dataset quality control.
3) **Execution on Target Device and Latency Computation:** On the user-defined target hardware, inference is performed (see ⓓ in Fig. 5). For every sampled model, we record the latency 150 times. The slowest and the fastest 20% of the inferences are discarded, and average is computed of the remaining 60%. As a quality control measure for our dataset, the latencies of the reference models are analyzed. Using reference models, we ensure the quality

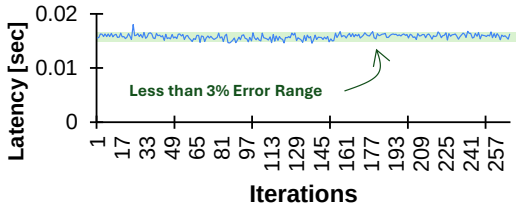


Fig. 6. Variance Plot of Reference models. Most of reference model instances fall within the 3% error boundary. The outliers are removed to improve the dataset quality.

of the dataset. This enables us to record the variance of our dataset, and identify outliers. If the variance is less than 3% (Fig. 6), the dataset passes the quality assurance; otherwise, it fails and the execution is performed again.

4) **Encoding Sampled Model Architectures:** Architectures of DNNs can be represented in various forms. Model architecture configurations after ❷ in Fig. 5, get encoded into vectors (see ❸ in Fig. 5). Let's say that a specific encoding scheme (η) is a function that translates an architecture to a vector.

$$\mathbf{z_{arch}} = \eta(\text{Architecture})$$

These representations depend largely on the type of architectural space the user defines. Typically, one hot vectors can be utilized to reflect any DNN. But such a representation gives rise to vectors that are extremely lengthy and sparse. Several state-of-the-art techniques employ feature encoding to shorten and reduce the sparsity of the vectors while aiming to maintain feature information [16] [17] [18]. Another common technique is to take mean and standard deviation of features [11] (Fig. 7d). Although both of these embeddings significantly shorten the vector as compared to one-hot vectors, they do not completely solve the problem. Statistical encoding (Fig. 7d) make the vectors too dense, causing loss of crucial information, and disabling the predictor to efficiently learn all parameters. On the other hand, Feature encoding (Fig. 7d) does detail out features of an architecture, the vector is still quite lengthy and sparse.

We propose an encoding scheme, Feature Combination Count (FCC) Encoding, that is not only significantly shorter than one-hot vectors, but also is not too dense (Fig. 7c). FCC first makes combinations of all possible features of a block, and then counts the number of instances each combination is used in the given model. This allows the vectors to retain enough information about any specific model's features for the predictor to perform optimally, while also getting rid of sparsity. We also experimented with Feature Count Encoding (Fig. 7c). This embedding counts the number of occurrences of each feature. Granted, this encoding retained a healthy amount of feature information; however, the vectors were still lengthy and sparse.

The dataset D is made up of these architecture-latency pairs obtained (Fig. 5e).

D. Predictor Training and Evaluation

The latency predictor is an MLP-based regression model designed to predict the latency of any given model architecture within a specific sample space for the target hardware. Primarily, we train our MLP on dataset D and perform evaluation on each of the bins (see ❸ in Fig. 5). This evaluation is based on the accuracy threshold, Acc_{TH}, defined by the user. If all the bin's accuracy, Acc_{i}, is greater than the threshold, Acc_{TH}, the predictor meets the evaluation criteria. If not, all the accuracies of the bins, Acc_{Bins}, are noted. The framework proceeds to dataset extension, after which the predictor is retrained and re-evaluated.

Algorithm 1: Pseudo-code for Dataset Extension

Input: SS: Entire Sample Space
Input: SS_{I}: Initial Sampling Space
Input: N_{Step}: Total number of additional training samples to select
Input: N_{I}: Number of Initial Samples
Input: Acc_{TH}: Accuracy threshold for bin categorization
Input: strategy: Sampling strategy, "$random$" or "$balanced$"
Input: w_1 = Weight of Below Threshold Bins
Input: w_2 = Weight of Above Threshold Bins
Output: Randomly sampled additional data $SS_{\text{I,rand}}$
Output: Weighted sampled additional data $SS_{\text{I,weighted}}$

if *strategy* = *random* **then**
  Randomly select N_{Step} samples from SS to form SS_{rand};
  Append SS_{rand} to SS_{I}
  $SS_{\text{I,rand}} \leftarrow SS_{\text{I}} + SS_{\text{rand}}$
  return $SS_{\text{I,rand}}$
end
else if *strategy* = *balanced* **then**
  Initialize empty list of No.of samples per bin: $\mathbf{N_{Bins}}$
  /* Separate bins N_{Bin_i} into two groups */
  $Below_{\text{Acc}_{\text{TH}}} \leftarrow$ Bins with accuracy $< Acc_{\text{TH}}$
  $Above_{\text{Acc}_{\text{TH}}} \leftarrow$ Bins with accuracy $\geq Acc_{\text{TH}}$
  for $Below_{Acc_{TH}}$ *and* $Above_{Acc_{TH}}$ **do**
    $N_{\text{B}_{\text{TH}}} = \text{len}(Below_{\text{Acc}_{\text{TH}}})$
    $N_{\text{A}_{\text{TH}}} = \text{len}(Above_{\text{Acc}_{\text{TH}}})$
  end
  /* Calculate samples per N_{Bin_i} */
  $N_{\text{norm}} = w_1 \times N_{\text{B}_{\text{TH}}} + w_2 \times N_{\text{A}_{\text{TH}}}$
  $N_{\text{Below}_{\text{Acc}_{\text{TH}}}} \leftarrow N_{\text{Step}} \times \frac{w_1}{N_{\text{norm}}}$
  $N_{\text{Above}_{\text{Acc}_{\text{TH}}}} \leftarrow N_{\text{Step}} \times \frac{w_2}{N_{\text{norm}}}$
  $N_{\text{Bins}} \leftarrow \lceil N_{\text{Below}_{\text{Acc}_{\text{TH}}}} \rceil + \lceil N_{\text{Above}_{\text{Acc}_{\text{TH}}}} \rceil$
  for i *in* N_{Bin_i} **do**
    Sample N_{Bin_i} from SS
    Append each sampled N_{Bin_i} to SS_{weighted}
  end
  Append SS_{weighted} samples into SS_{I}
  $SS_{\text{I,weighted}} \leftarrow SS_{\text{I}} + SS_{\text{weighted}}$
  return $SS_{\text{I,weighted}}$;
end

E. Dataset Extension

If the predictor fails to meet the evaluation criteria, the framework looks to extend the dataset for re-training and re-evaluation of the predictor (see ❹ in Fig. 5). This extension is executed using **Algo. 1.**

The user defines total number of additional training samples to select, N_{Step}. If the sampling strategy is $random$, N_{Step} samples are randomly selected from the entire sample space and appended to the initial sample space. These samples then proceed and undergo the usual dataset generation pipeline where first reference models are added, then the samples are encoded and executed on a target device. These new architecture-latency pairs are appended to the previous dataset D. The MLP is then re-trained and re-evaluated. This loop repeats until all the bins' accuracies, Acc_{i}, meet the threshold accuracy, Acc_{TH}.

If the user had defined the sampling strategy as $balanced$, a $weighted$ approach with regards to the bins is taken. The bins, N_{Bins} are separated into two categories: Below Accuracy Threshold ($Below_{\text{Acc}_{\text{TH}}}$), and Above Accuracy Threshold

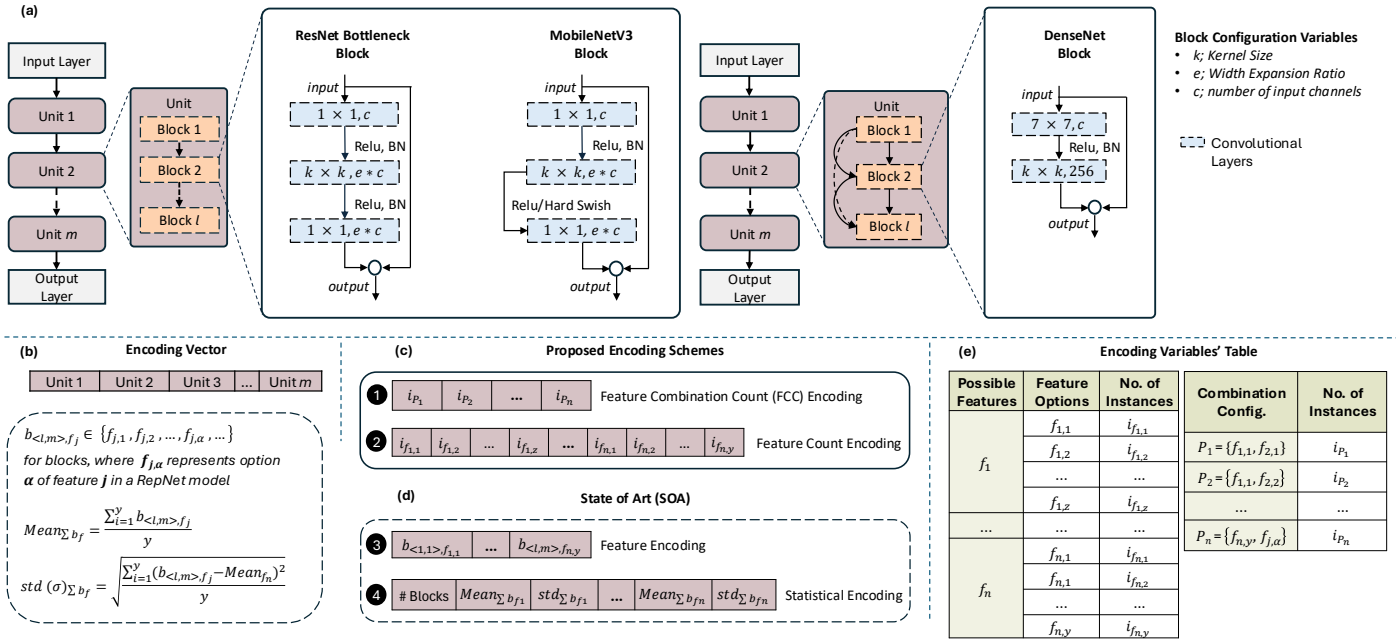


Fig. 7. (a) General representation of ResNet, MobileNetV3, and DenseNet supernets. (b) General representation of encoding, illustrating concatenation of unit-level features. (c) Our proposed encoding schemes. (d) State-of-the-art encoding schemes (e) Table containing information of variables used in the encoding schemes.

TABLE I
SUPERNET ARCHITECTURES AND ASSOCIATED HYPERPARAMETERS

Variable	ResNet	MobileNetV3	DenseNet
Stage Width List	$[2^8, 2^9, 2^{10}, 2^{11}]$	$[2^4, 2^5, 2^6, 2^7]$	N/A
# of Units	4	4	5
# of Blocks per Unit.	{1, 2, ..., 7}	{1, 2, ..., 7}	{1, 2, ..., 20}
Kernel Size Options	{3, 5, 7}	{3, 5, 7}	{1, 3, 5, 7, 9}*
Width-Expansion Ratio Options	{1/2, 2/3, 1}	{1/2, 2/3, 1}	N/A
# of Archs.	8.38×10^{26}	8.38×10^{26}	10^{10}

* For DenseNet, each kernel size is applied to all the blocks in a unit.

($Above_{\text{Acc}_{\text{TH}}}$). For each category, number of samples are calculated by applying weights w_1 and w_2, respectively, to N_{Step} for each category of bins. This can be seen in **Algo. 1**. This creates a bias in the dataset towards the below accuracy threshold. When re-training, the MLP learns their patterns as well and eventually crosses the accuracy threshold, Acc_{TH}.

III. Results and Discussion

A. Experimental Setup

In this section, we describe the supernet architectures, regression models, and target hardware devices used in our experiments. We used three different supernet architectures: ResNet, MobileNetV3, and DenseNet. The associated hyperparameters are presented in Table I.

Regression Model: We used an MLP-based regression model consisting of three fully-connected layers, each with a hidden dimension size of 64. The model is trained using Mean Squared Error (MSE) as the loss function and the Adam optimizer, with a learning rate of 0.01 and a weight decay factor of 10^{-4}. Unless mentioned otherwise, the predictor is trained on 8,000 samples and tested on 4,000 samples.

Target Hardware: We evaluated our framework on multiple devices presented below:

- **CPUs:** AMD Ryzen Threadripper 5975, Raspberry Pi 4
- **GPUs:** NVIDIA RTX 4090, NVIDIA RTX 3080 Max-Q

B. Effectiveness of Proposed Encoding vs SoTA

Fig. 8 shows scatter plots of predicted latencies against actual latencies for ResNet and DenseNet. For this experiment, we trained the latency predictor on 8,000 and 20,000 samples using different encoding schemes for ResNet and DenseNet supernets. The plots show a comparison of the latency predictor's performance using different encoding schemes (proposed FCC vs. Statistical), and a different (Look-up Table-based) surrogate model. These results are further summarized in Fig. 9 where it can be seen that the MLP trained on FCC encoding comprehensively outperforms Look-up Table and Statistical Encoding methods. The predictor trained for ResNet supernet on FCC encoded data gave average accuracies of 97.6%, and 97.8% for 8,000 and 20,000 samples, respectively. Conversely, statistical encoding gave average accuracies of 85.8% and 83.1% for 8,000 and 20,000 samples, respectively. Performing bias correction using a linear regression model significantly improved the Lookup Table predictions with an average accuracy of 83.9% for ResNets. Similarly, FCC gave 99% accuracy, while Lookup Table with Bias Correction gave 97% accuracy for DenseNet.

C. FCC Encoding Effectiveness Across Devices

Figure 10 displays the performance of the predictor on the different target devices for the DNNs in Fig. 7**(a)**. Again, for each device we present a comparison of our proposed FCC and FC encodings with SoTA Statistical Encoding. Training size samples for RTX 4090 was 8,000, 5,000 for AMD CPU and RTX 3080Ti, and 1,200 for Raspberry Pi 4. It can be seen that for each of the three DNNs, DenseNet, MobileNetV3, and ResNet, FCC and FC outperform Statistical Encoding on most devices. For ResNet super-net(Fig 10(a), the average accuracies across the 4 devices listed were 97%, 88%, 93%, and 99% for proposed FCC encoding and 90%, 84%, 82% and 99% for FC encoding. While the averages for statistical encoding were 85%,

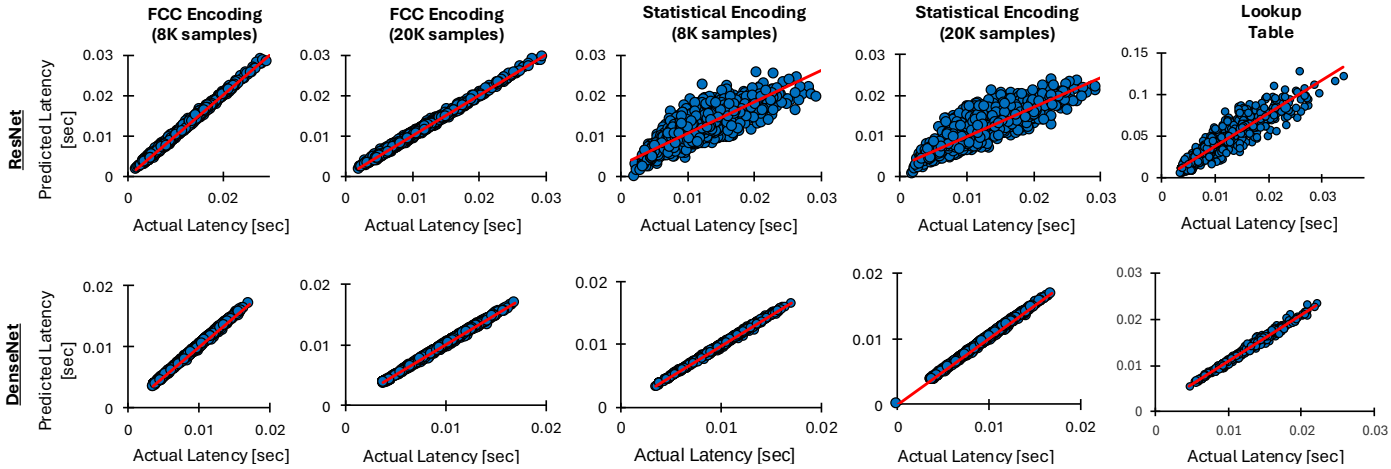


Fig. 8. Scatter plots illustrating a comparison between FCC-based MLP with Statistical Encoding-based MLP and Lookup-table for ResNet(Top-row) and DenseNet(Bottom-row). These results are for NVIDIA RTX 4090.

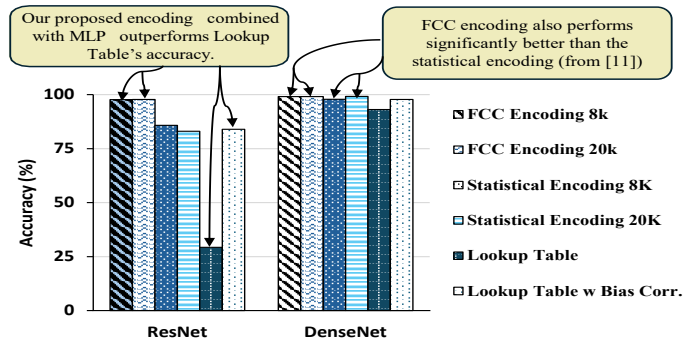


Fig. 9. Average accuracies' comparison for the cases presented in Fig 8.

83%, 71%, and 98%, respectively. For MobileNetV3 super-net (Fig. 10(b), the average accuracies across the 3 devices listed were 97%, 99%, and 99% for proposed FCC encoding, and 98%, 99% and 99% for FC encoding, while the averages for statistical encoding were 96%, 97%, and 97%, respectively. For DenseNet super-net (Fig. 10(c), the average accuracies across the 3 devices listed were 99%, 94%, and 94% for proposed FCC encoding and 99%, 96% and 95% for FC encoding, while the averages for statistical encoding were 91%, 96%, and 95%, respectively.

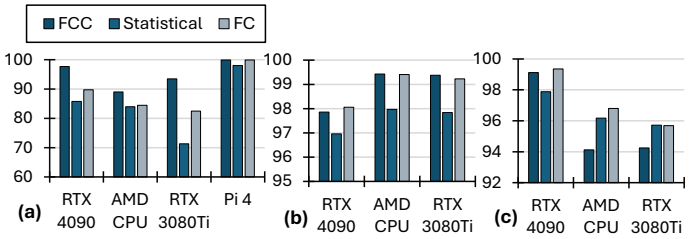


Fig. 10. Showcasing effectiveness of the proposed encoding schemes across different target devices: NVIDIA RTX 4090, AMD Ryzen Threadripper 5975, NVIDIA RTX 3080 Max-Q, and Raspberry Pi 4. The three plots are for (a) ResNet, (b) MobileNetV3, and (c) DenseNet.

D. Random and Balanced Accuracy vs Training Size

Figure 11 shows latency prediction results for *random* and *balanced* sampling strategies. For this experiment we computed latency of a ResNet super-net on NVIDIA RTX 4090. We defined N_I as 300 and set N_{Step} to be 100 and used ESM methodology for training the predictor. The plot clearly showcases that *balanced* sampling strategy converges much more earlier than *random* sampling strategy. *Balanced* sampling strategy converged after 3 iterations with 500 samples, while *random* sampling strategy took 37 iteration to converge with 4000 samples.

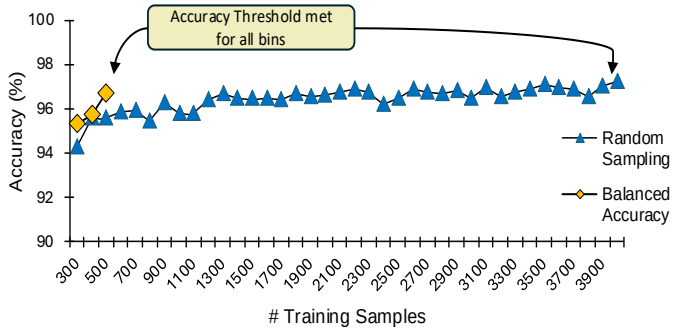


Fig. 11. Comparison between Random and Balanced Sampling Strategy for Latency Predictor tested on ResNet super-net on NVIDIA RTX 4090.

IV. Conclusion

In conclusion, this paper highlights the strengths and weaknesses of various surrogate models along with different encoding schemes. Our proposed framework, ESM, along with our novel encoding scheme, FCC, produce results that outperform current State-of-the-Art surrogate models and encoding schemes. Moreover, we provide clear analysis of how the ESM is scalable to a wide array of devices such as GPUs, CPUs, and embedded devices. Additionally, we also present how FCC can be scaled to various DNNs. Based on the insights gained from the analysis, we present a holistic framework that enables accurate, efficient latency prediction using HW-NAS for a variety of target hardware and neural networks, while adhering to user-defined constraints.

Acknowledgment

This work was partially supported by the NYUAD Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010.

REFERENCES

[1] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, no. 1, p. 7068349, 2018.

[2] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.

[3] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[4] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[5] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[6] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "Hardware-aware neural architecture search: Survey and taxonomy." in *IJCAI*, 2021, pp. 4322–4329.

[7] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 285–300.

[8] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[9] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 10 734–10 742.

[10] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2078–2087.

[11] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," *arXiv preprint arXiv:2005.14187*, 2020.

[12] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[13] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[14] L. Mauch, S. Tiedemann, J. A. Garcia, B. N. Cong, K. Yoshiyama, F. Cardinaux, and T. Kemp, "Efficient sampling for predictor-based neural architecture search," *arXiv preprint arXiv:2011.12043*, 2020.

[15] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda *et al.*, "Fbnetv3: Joint architecture-recipe search using predictor pretraining," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 276–16 285.

[16] S. Hassantabar, X. Dai, and N. K. Jha, "Curious: Efficient neural architecture search based on a performance predictor and evolutionary search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4975–4990, 2022.

[17] Z. Li, M. Paolieri, and L. Golubchik, "Inference latency prediction at the edge," *arXiv preprint arXiv:2210.02620*, 2022.

[18] D. M. Mendoza and S. Wang, "Predicting latency of neural network inference," 2020.

[19] X. Ning, Y. Zheng, T. Zhao, Y. Wang, and H. Yang, "A generic graph-based neural architecture encoding scheme for predictor-based nas," in *European Conference on Computer Vision*. Springer, 2020, pp. 189–204.

ZION: A Practical Confidential Virtual Machine Architecture on Commodity RISC-V Processors

Jie Wang*, Juan Wang*†, Yinqian Zhang‡
*Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education,
School of Cyber Science and Engineering, Wuhan University, Wuhan, China
†Zhongguancun Laboratory
‡Southern University of Science and Technology
{iwangjye, jwang}@whu.edu.cn, yinqianz@acm.org

***Abstract*—Trusted Execution Environments (TEEs) provide robust hardware-based isolation to mitigate data breaches and privacy risks. Confidential Virtual Machines (confidential VMs or CVMs) extend these capabilities by using VMs as their execution abstraction, offering superior compatibility over process-based TEEs like Intel SGX. The rising demand for Confidential VMs has spurred innovations from major chip manufacturers, such as AMD SEV, Intel TDX, and Arm CCA, and their integration into leading cloud platforms, including AWS, Azure, and Google Cloud. On the RISC-V platform, however, existing TEE architectures rely on process-level abstractions or custom hardware, leading to limited compatibility and scalability.**

This paper presents ZION, a confidential VM architecture for commodity RISC-V hardware that operates without custom extensions. ZION ensures security, flexibility, and efficiency through a short-path *CVM mode* and a secure vCPU mechanism for protecting and efficiently updating vCPU states, enhancing context-switching performance. It combines Physical Memory Protection (PMP) with paging for scalable memory isolation, employs a hierarchical memory structure for efficient management, and introduces a split-page-table-based mechanism for secure memory sharing with virtio devices. Evaluations show ZION achieves under 5% overhead in real-world applications, demonstrating its practicality.

***Index Terms*—Confidential VM, RISC-V, Trusted Execution Environment**

I. INTRODUCTION

Trusted Execution Environments (TEEs) provide robust hardware-based isolation, making them increasingly crucial in addressing the rising incidents of data breaches and privacy violations [1], [2]. Among TEE technologies, Confidential Virtual Machines (confidential VMs or CVMs) stand out by using VMs as their execution abstraction. This approach offers greater compatibility and practicality compared to process-based TEEs like Intel SGX [3].

Driven by these advantages, confidential VM technology has seen significant development and adoption. Leading chip manufacturers have introduced their own confidential VM architectures, such as AMD SEV [4], Intel TDX [5], and Arm CCA [6]. Similarly, major cloud providers, such as Amazon AWS [7], Microsoft Azure [8], and Google Cloud [9], have integrated confidential VMs to offer secure and privacy-focused services for tenants with confidential workloads.

Confidential VMs on RISC-V are also gaining increasing attention and adoption, driven by the expanding role of RISC-V across diverse computing domains. Initially, RISC-V established its presence in low-performance Internet of Things (IoT) devices, leveraging its open-source architecture and flexibility to support a wide range of applications. However, RISC-V [10] is now transcending its traditional focus, making notable progress in high-performance cloud computing environments that demand robust, secure, and efficient solutions. This evolution is fueling a growing need for confidential VM technology, as organizations prioritize data privacy and secure computation in these advanced settings.

However, practical confidential VM architectures are still lacking on the current RISC-V platform. Sanctum [11], Keystone [12], and Penglai [13], the prominent RISC-V TEE architectures, all use process-level abstractions, necessitating modifications to user applications to be compatible with their TEEs. In contrast, confidential VM architectures designed to run on the RISC-V platform face issues with hardware compatibility. For example, CURE [14] and VirTEE [15] offer VM enclaves capable of running confidential VMs via hardware extensions. However, their reliance on custom hardware introduces hardware compatibility issues—they cannot run on commodity RISC-V processors. Another initiative, the confidential VM extension (CoVE) [16] proposed by RISC-V TEE TaskGroup (AP-TEE TG) [17], is still under development [18]. Considering the long delay from design to actual adoption of hardware security primitives, the availability of CoVE in the near future remains uncertain.

In addition to hardware compatibility, the current confidential VM architecture of the RISC-V platform also has deficiencies in flexibility and scalability. In terms of flexibility, CURE [14] and VirTEE [15] implement region-based memory isolation, requiring the physical addresses of the memory isolation area to be continuous. This design struggles to meet the flexible memory isolation needs of confidential VM, making dynamic expansion difficult and causing significant memory fragmentation. In terms of scalability, due to the

Corresponding author: Juan Wang

This work was supported in part by the National Key R&D Program of China under Grant No. 2023YFB4503902, the Smart Grid-National Science and Technology Major Project (No. 2024ZD0803000), the Major Program (JD) of Hubei Province (No. 2023BAA027), and the Key R&D Projects in Hubei Province under Grant No. 2023BAB165.

resource limitations of the security primitives of custom hardware extensions, CURE and VirTEE can only support 13 concurrent VM enclaves, which is difficult to meet the large-scale concurrency requirements of cloud platforms. These challenges have led us to pose the following research question: *Can we design a secure, efficient, and flexible confidential VM architecture using the commodity RISC-V processors?*

In this paper, we present ZION, a confidential VM architecture built on the commodity hardware of the RISC-V platform. ZION introduces several innovative designs to ensure security, flexibility, and efficiency. It introduces a short-path *CVM mode* and combines a secure vCPU with a shared vCPU interaction mechanism, enabling secure and efficient protection and updates of vCPU states. This ensures both the security and performance of context switching between confidential VMs. For memory isolation, ZION combines the Physical Memory Protection (PMP) hardware isolation primitive with paging to create a flexible and scalable memory isolation mechanism. Additionally, ZION incorporates a hierarchical memory structure and allocation strategy, significantly improving memory management efficiency for confidential VMs. Furthermore, ZION introduces a secure and efficient memory sharing mechanism based on split page tables, enabling support for virtio devices of confidential VMs. Evaluations show that ZION incurs less than 5% performance overhead in most real-world applications.

In sum, this paper makes the following contributions:

- We design a practical confidential VM architecture atop commodity RISC-V processors. This design does not require any hardware security extensions, ensuring compatibility with existing RISC-V hardware.
- We propose a series of optimizations for confidential VMs, including isolation mode, vCPU state protection and updates, memory isolation and management, and memory sharing, which ensure the flexibility, security, and efficiency of the ZION architecture.
- We implement a prototype of ZION and conduct comprehensive performance evaluation, which show that the overhead introduced by ZION is under 5% for most real-world applications.

II. BACKGROUND

RISC-V is an open, patent-unencumbered reduced instruction set computing (RISC) architecture [19]. RISC-V processors can be customized, implemented, and utilized by any entity. In addition to its clean, modular design, RISC-V includes specific security features, such as:

- **Physical Memory Protection:** PMP serves as a physical memory isolation mechanism, allowing software in machine mode (M mode) to control the memory access permissions of software in lower privilege modes. Each hardware thread has a set of PMP entries that define the address, size, and access permissions for multiple memory regions.
- **Trap Delegation:** By default, all traps (including interrupts and exceptions) on the RISC-V platform are handled in M mode. To improve the efficiency of trap handling, RISC-V introduces a trap delegation mechanism, which allows traps to be directly handled in lower privilege modes.
- **Hypervisor Extension:** To enable virtualization, RISC-V introduces the hypervisor extension, which extends supervisor mode to hypervisor-extended supervisor mode (HS mode). This extension also adds virtual supervisor (VS) and virtual user (VU) modes to support VMs. Hypervisor extension introduces new control and status registers and instructions to facilitate the execution of VMs and assist the hypervisor in managing them. Additionally, RISC-V supports memory virtualization through two-stage address translation in its virtualization modes.

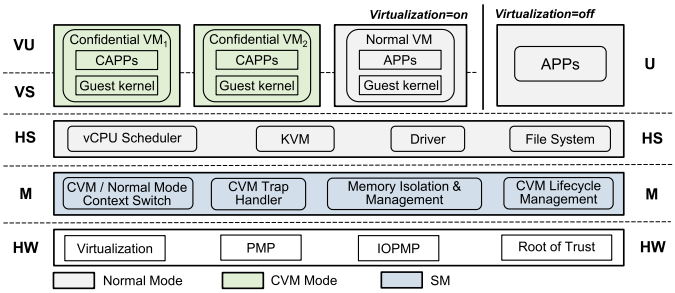


Fig. 1: The architecture of ZION.

III. OVERVIEW

A. System Architecture

The system architecture of ZION, as shown in Figure 1, introduces two new modes: *Normal mode* and *CVM mode*. Mode switching between the two is managed by the Secure Monitor (SM). *Normal mode* is responsible for running the non-trusted components, including the hypervisor, upper-layer applications, and normal VMs. The hypervisor operates in HS mode, which encompasses the scheduler, device drivers, file system, and the KVM. Meanwhile, QEMU runs in U mode, providing simulated device support for confidential VMs.

ZION is dedicated to running confidential VMs and operates in virtualized environments. The kernel of the confidential VM runs in VS mode, while its applications run in VU mode. The implementation and security of ZION are ensured by the SM, which operates at the highest privilege level (the M mode) and serves as the Trusted Computing Base (TCB) for the system. The SM uses hardware security primitives (e.g., PMP, IOPMP, trap delegation) to construct ZION, providing secure isolation and protection for the vCPUs and memory of confidential VMs. In addition, the SM handles memory management in ZION, including allocation, reclamation, and secure memory sharing. To facilitate collaboration between *Normal mode* and *CVM mode*, SM offers different Environment Call (ECALL) interfaces. These interfaces allow the hypervisor in *Normal mode* to control the lifecycle of confidential VMs, including initialization, operation, and suspension. Additionally, confidential VMs in ZION can use these ECALL interfaces to access functions such as retrieving measurement reports and platform random numbers.

B. Threat Model

ZION's threat model assumes the trustworthiness of underlying hardware mechanisms, including virtualization, trap delegation, PMP, and IOPMP. The security model assumes the SM is reliable and correctly implements the security mechanisms described in this paper. However, ZION does not trust privileged software beyond the SM and assumes the hypervisor could be fully compromised, including critical functions like vCPU scheduling and memory management. It is also assumed that the confidential VM does not intentionally leak sensitive data or code.

Physical attacks, such as bus snooping [20], cold boot attacks [21], and fault injection [22], [23], are beyond the scope of our research, as they require additional hardware support (e.g., a memory encryption engine) for mitigation. Similarly, cache side-channel attacks [24] and transient execution attacks [25], [26] fall outside our threat model, though existing defense mechanisms [27], [28] can be integrated into ZION's design. Like other TEE architectures, ZION does not address denial-of-service attacks.

IV. ZION DESIGN

A. Short-path CVM Mode

Most current confidential VM architectures, such as ARM CCA [6], TwinVisor [29], and CoVE [16], follow a similar model. These designs rely on the monitor to handle CPU security state transitions and introduce a thin secure hypervisor through horizontal isolation technologies, such as the TEE Security Manager (TSM) in CoVE. The secure hypervisor, which is much smaller than the normal hypervisor, provides essential security functions for confidential VMs. However, this approach increases the context switching path, resulting in performance degradation.

In its isolation mode, ZION adopts a design based on a virtual security processor, which divides the CPU into *Normal mode* and protected *CVM mode* using time-sharing multiplexing techniques. Unlike other designs, ZION centralizes execution state switching and security management in the SM, allowing the switch between *CVM mode* and *Normal mode* to require only a single privilege level switch. This approach reduces the overhead caused by multiple privilege level switches, as seen in traditional confidential VM architectures.

While ZION avoids introducing a secure hypervisor, this raises the risk that traps from confidential VMs could be captured by the normal hypervisor, potentially leading to security issues like leakage of sensitive register states. To mitigate this risk, ZION implements a trap delegation control mechanism that transfers trap handling rights to the SM. The delegation criteria are as follows: 1) Traps that can be processed by the confidential VM are delegated to VS mode and handled by the confidential VM kernel; 2) Traps that cannot be processed directly are sent to the SM for handling. This design improves context switching efficiency while maintaining the security of the confidential VM.

B. vCPU State Protection and Update

Unlike normal VMs, where the vCPU state is entirely controlled by the hypervisor, the vCPU state of confidential VMs must be protected from tampering or theft by untrusted hypervisors for security reasons. However, certain confidential VM functionalities, such as MMIO, still require assistance from the hypervisor. The challenge lies in ensuring the security of the confidential VM's vCPU state while facilitating efficient state transfer between the confidential VM and the hypervisor.

To address this, ZION introduces a secure and efficient vCPU state protection and update mechanism, combining secure and shared vCPUs. ZION allocates memory within the SM to create a secure vCPU structure. This structure stores the general-purpose and control register states of each vCPU of the confidential VM, ensuring the security of the vCPU state. To accelerate state transfer, ZION establishes a shared vCPU structure in the hypervisor, which is used for fast exchange of vCPU states between the hypervisor and the SM.

During the transition between *CVM mode* and *Normal mode*, the SM stores the register states in either secure vCPU or shared vCPU based on the type of exception or interrupt that causes the confidential VM to exit. For example, if a load instruction triggers a confidential VM exit, the SM stores the trap-related register states, such as *htinst*, in the shared vCPU to allow the hypervisor to parse the exception and retrieve the target register of the load instruction. Other register states are stored in the secure vCPU to maintain isolation and protection.

After the hypervisor handles the exception, it writes the required values into the shared vCPU registers. When the confidential VM is resumed, the SM not only restores the register states from the secure vCPU but also retrieves necessary values from the shared vCPU. To protect against TOCTOU (Time-of-Check to Time-of-Use) attacks [30], ZION adopts a Check-after-Load mechanism from TwinVisor, performing security checks after reading the shared vCPU register values to prevent tampering by the malicious hypervisor [29].

C. Flexible and Scalable Memory Isolation

A straightforward approach to memory isolation is to allocate contiguous physical memory directly to confidential VMs and utilize the PMP to isolate this memory region. While this method ensures memory isolation, it presents two key challenges: first, the limited number of PMP entries (typically 16) restricts the number of concurrent confidential VMs; second, this memory pre-allocation strategy requires resources to be allocated in advance based on the expected memory size when creating the confidential VM. This approach not only weakens the cloud platform's ability to overcommit memory, but also leads to memory fragmentation, while limiting the confidential VM's ability to dynamically expand memory.

To enhance the flexibility and scalability of memory isolation, ZION combines PMP with paging. This approach leverages PMP to enforce memory isolation between *CVM mode* and *Normal mode*. Initially, the SM configures PMP to create a secure memory pool within normal memory, designated for confidential VMs. Before transitioning to *CVM mode*, the SM

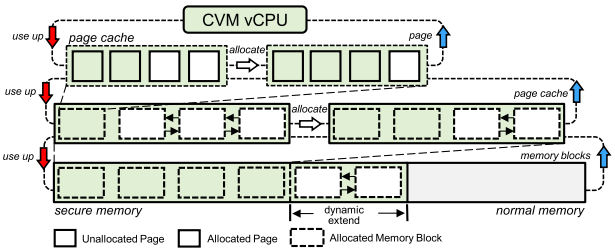


Fig. 2: Hierarchical memory allocation for confidential VMs.

updates the PMP configuration register (*pmpcfg*) to grant the confidential VM access to this secure memory. During hypervisor operation, the SM disables access to the secure memory pool, ensuring that untrusted hypervisors cannot modify or corrupt the protected memory.

For memory isolation between confidential VMs, ZION relies on Stage-2 page table isolation. During the creation and page fault handling of a confidential VM, the SM ensures that the memory allocated to the confidential VM is not shared with other confidential VMs. Only the SM has authority over the page table of the confidential VM, ensuring that the secure memory where the confidential VM's page table is stored is not mapped into the address space of other confidential VMs. As a result, even if a confidential VM is compromised, it cannot access or modify the memory of other confidential VMs due to the restrictions enforced by the page table.

To safeguard against potential Direct Memory Access (DMA) attacks from malicious peripherals, the secure memory area is also protected by the IOPMP [31], which controls peripheral access in addition to the CPU-level protection provided by PMP. Furthermore, to prevent untrusted hypervisors from launching page table-based attacks, such as side-channel or controlled-channel attacks [32], the SM configures page tables for confidential VMs within the secure memory pool. This prevents the hypervisor from tampering with or interfering with the confidential VM's page table.

D. Hierarchical Memory Management

To efficiently manage the secure memory pool of *CVM mode*, ZION introduces a hierarchical secure memory structure and employs targeted memory allocation strategies at various levels. When a privileged user registers contiguous physical memory with the SM, the SM divides the memory segment into smaller secure memory blocks (default size of 256KB) and links these blocks in a bidirectional circular linked list. Each secure memory block consists of several contiguous memory pages and contains pointers to the previous and next blocks. The blocks are ordered by address, with allocation prioritized from the head of the linked list.

Figure 2 illustrates the memory allocation process for *CVM mode*, which is divided into three stages. In the first stage, the SM attempts to allocate a memory page directly from the page cache associated with the vCPU of the confidential VM in which the page fault occurs. If free memory pages are available in the page cache, one is allocated directly; otherwise, it enters the second stage of allocation. In this stage, the SM allocates a new secure memory block from the head of the linked list and associates it with the vCPU experiencing the page fault. This new block serves as the page cache for that vCPU, ensuring each vCPU has an independent cache. In the third stage, when the secure memory pool nears depletion, the SM requests the hypervisor to expand the secure memory pool. The expansion strategy includes applying for a new contiguous physical memory area to be registered as a secure memory pool or expanding an existing memory pool.

This hierarchical memory allocation model significantly enhances memory management efficiency for confidential VMs. In the first stage, the availability of page cache allows most allocation requests to be fulfilled quickly, with the vCPU-specific cache supporting efficient memory allocation while avoiding the performance loss from concurrent locking. In the second stage, secure memory blocks are efficiently managed through the linked list, allowing allocation with a time complexity of $O(1)$. Since the first two stages are entirely controlled by the SM, they avoid the performance overhead associated with switching to *Normal mode*. Although the third phase requires the assistance of the hypervisor and introduces operations including memory expansion, mode switching, and TLB flushing, its impact on overall performance is minimal since most allocation requests can be handled in the earlier stages.

E. Split Page Table-Based Memory Sharing

To prevent the hypervisor from tampering with the confidential VM's page table, we assign page table management to the SM by default. However, this approach incurs performance overhead when supporting shared memory.

In the unoptimized shared memory establishment process, the hypervisor first allocates memory and creates a mapping, then synchronizes this mapping with the SM. After the SM verifies the request, it maps the confidential VM's address space to the physical memory, establishing the shared memory. However, this synchronization introduces performance overhead and consistency issues, particularly during page swap operations in *Normal mode*. If synchronization is delayed, it could threaten system availability and security. On the other hand, implementing a synchronization mechanism would increase system complexity and degrade performance.

To address these issues, we propose a shared memory scheme based on a split page table design. The Guest Physical Address (GPA) space of the confidential VM is divided into two regions: a private address space, secured by the SM for storing confidential code and data, and a shared address space, managed by the hypervisor for memory sharing. The shared address space maps directly to normal memory. The confidential VM's root page table includes entries for both secure and shared page tables, enabling access to memory from either. In contrast, the hypervisor's root page table contains only shared page table entries, restricting its access to normal memory and preventing modifications to the private memory mappings of the confidential VM. This design ensures that the hypervisor cannot use the shared page table it controls

to access the secure memory of the confidential VM, thereby maintaining memory security.

Since the hypervisor directly manages the shared page table, updates to shared memory can bypass the SM, eliminating the need for complex synchronization and reducing performance overhead. The update process affects only the shared address space, streamlining the operation and minimizing synchronization burdens. As a result, the split page table-based approach enhances both the efficiency and security of shared memory management while resolving related performance challenges.

V. Evaluation

A. Implementation & Experimental Setup

We integrated the SM with OpenSBI and made modifications to support our design. We also adapted the host's KVM module for compatibility and added the ECALL interfaces for calling SM. Additionally, we made minor adjustments to the confidential VM kernel to support shared memory requests.

To evaluate the performance of ZION, we conducted experiments on the Genesys2 FPGA development board, which simulated four 64-bit Rocket cores with hypervisor extensions, running at 100 MHz and equipped with 1 GB of memory. The host ran Linux (kernel version 5.19.16), and the virtualization stack used customized QEMU and KVM. The OS in the confidential VM matched that of the host. To support virtio devices, the confidential VM enabled SWIOTLB support and configured its buffer as shared memory.

B. CVM Mode Switching Performance

1) Optimization brought by shared vCPU mechanism: We evaluated the context switching time overhead of confidential VMs with and without shared vCPU state update support. The CVM entry and exit, triggered by MMIO exceptions, were tested 200 times. For CVM entry, the switching time without shared vCPU support was 5,293 cycles, while with shared vCPU support, it was reduced to 4,191 cycles, reflecting a performance improvement of 20.8%.

A similar improvement was observed in the CVM exit test. The switching time without shared vCPU support was 3,267 cycles, compared to 2,524 cycles with shared vCPU support, resulting in a performance improvement of 22.74%. These results demonstrate that ZION's shared vCPU state update mechanism significantly enhances context switching performance in *CVM mode*.

2) Optimization brought by short-path CVM mode: To assess the performance impact of ZION 's short-path design on *CVM mode* switching, we developed a simple secure hypervisor in the host system. This setup allowed us to compare the performance of long-path and short-path *CVM mode* context switching. The long-path *CVM mode* introduces an additional privilege layer conversion during context switching. At CVM entry, the execution flow jumps from the host to the SM, then to the secure hypervisor, and finally to the confidential VM. During CVM exit, the flow moves first from the *CVM mode* to the secure hypervisor, then from the hypervisor to the SM, and finally back to the host.

TABLE I: Execution cycles for RV8 Benchmarks in various environments. The baseline is indicated by *, and all values are in 10^9 cycles.

Benchmark	Normal VM*	Confidential VM (%)	
aes	6.312	6.498	(+2.95)
bigint	8.965	9.210	(+2.73)
dhrystone	4.144	4.264	(+2.90)
miniz	25.412	25.900	(+1.92)
norx	3.905	4.014	(+2.79)
primes	19.002	19.347	(+1.81)
qsort	2.148	2.205	(+2.65)
sha512	3.947	4.063	(+2.93)
Average	-	-	+2.59

We tested CVM entry and CVM exit triggered by a timer interrupt 200 times under both the long-path and ZION 's short-path context switching. Since the timer interrupt does not involve vCPU state updates, this test isolates the impact of context switching alone. The results show that ZION 's short-path design significantly improves context switch performance. For CVM entry, the long-path *CVM mode* switching time is 7,282 cycles, while ZION 's short-path mode takes only 4,028 cycles, resulting in a 44.7% performance improvement. For CVM exit, the long-path switching time is 5,384 cycles, whereas ZION 's short-path mode takes just 2,406 cycles, yielding a 55.3% improvement.

It is important to note that in our test, the secure hypervisor was not fully isolated. As such, this test only partially reflects the performance penalty of the long-path mode. Once security measures such as microarchitecture state clearing are added to the secure hypervisor, these operations will likely have a more significant impact on the overall performance of the confidential VM.

C. Stage-2 Page Fault Handling Performance

In this section, we evaluated the page fault processing performance of the normal VM and the confidential VM. After both VMs were started, we ran a program that allocated continuous physical memory and performed write operations. For the normal VM, we directly recorded the stage-2 page fault processing time in KVM. For the confidential VM, we recorded the page fault processing time at different memory allocation stages in the SM.

The test results show that the average page fault processing time for the normal VM is 39,607 cycles. For the confidential VM, the processing time for page faults triggering only the first-stage memory allocation is 31,103 cycles. When the second-stage memory allocation is triggered, the processing time increases slightly to 34,729 cycles. However, when the third-stage memory allocation is triggered, the page fault processing time increases significantly to 57,152 cycles.

Since most page fault requests are handled during the first two stages, the average page fault processing time for the confidential VM is 31,449 cycles, which is only slightly higher than the first-stage allocation time. These results align with our design analysis and demonstrate the efficiency of ZION 's hierarchical memory management.

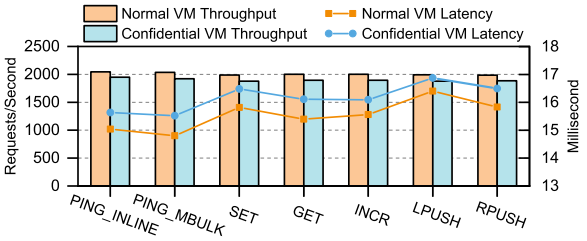


Fig. 3: Comparison of throughput and latency for various operations between normal VMs and confidential VMs.

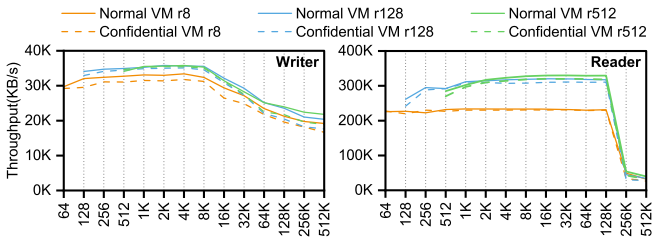


Fig. 4: Throughput performance for read and write operations at varying file sizes in normal and confidential VMs.

D. Macrobenchmarks

In this section, we evaluate confidential VM performance on real-world applications using CPU-intensive (RV8, Coremark), memory-intensive (Redis), and I/O-intensive (IOZone) workloads. Both normal and confidential VMs were configured with one vCPU, 256MB memory, and SWIOTLB enabled.

RV8 Benchmarks. The RV8 benchmarks include CPU-intensive programs such as encryption and decryption, mathematical operations, and sorting. We ran each program in the RV8 benchmark suite 20 times and calculated the average value for each. As shown in Table I, compared to normal VMs, the overheads of confidential VMs remained within 3%, with the average overhead measured at 2.59%.

CoreMark. CoreMark is another CPU-intensive benchmark. We ran it 20 times to obtain the average score. The test results indicate that the average score on the normal VM is 2,047.6, while the confidential VM scored 1,992.3, reflecting a 2.77% performance drop compared to the normal VM. This result aligns closely with the findings from our RV8 benchmark tests.

Redis Benchmark. The Redis benchmark tests the performance of the Redis in-memory database, which is itself a memory-intensive test. For this test, we conducted ten rounds of testing, executing 10,000 requests in each round. The test results are presented in Figure 3. The overhead of confidential VMs remains within a reasonable range when compared to normal VMs. In terms of throughput, confidential VMs show an average decrease of 5.3% compared to normal VMs. Regarding latency, confidential VMs experience an average increase of 4%.

IOZone. In this test, we compared continuous file read/write performance between a confidential VM and a normal VM across various file sizes (64KB–512MB) and record sizes (8KB, 128KB, 512KB). As shown in Figure 4, both write and read throughput are lower when the record size is small. For smaller files, the performance difference is minimal (under 5%), but as file sizes grow, the confidential VM's overhead increases, reaching up to 20% due to frequent I/O exits.

VI. Related Work

Sanctum [11], Keystone [12], Penglai [13], TIMBER-V [33], and SPEAR-V [34] are notable TEE implementations on RISC-V, all of which adopt the process model as their isolation abstraction. While effective, these process-based TEEs require significant porting efforts to support legacy applications, as they cannot run them directly. In contrast, Zion, a VM-based TEE, enables the execution of unmodified applications, offering greater practicality and ease of deployment.

Several VM-based TEEs also exist on the RISC-V platform, including CURE [14], VirTEE [15], and CoVE [16]. CURE supports multiple TEE abstractions, including user-space, kernel-space, sub-space, and VM enclaves, with the VM enclave enabling confidential VMs. Building on CURE, VirTEE introduces features such as live migration and secure I/O. However, both CoVE and VirTEE rely on hardware modifications, including enhancements to the CPU core and bus arbiter. These hardware dependencies limit their scalability, flexibility, and the number of parallel enclaves they can support. Zion, by comparison, operates without hardware modifications, offering greater scalability and flexibility, particularly in memory isolation and management. CoVE, developed by the AP-TEE TG [17], is a proposed confidential VM extension. It introduces a Memory Tracking Table (MMT) for controlling physical memory access and a lightweight secure hypervisor, the TSM, to isolate confidential VMs and provide essential security services. However, CoVE's specifications are still under development, and compatible hardware platforms are not expected to be widely available in the near term. In comparison, Zion leveraging RISC-V's mature hardware ecosystem, provides a practical alternative. It supports confidential VMs without requiring specialized hardware, making it a valuable solution even after CoVE's specifications and hardware become available.

On the Arm platform, works like Twinvisor [29] and VirtCCA [35] aim to support confidential VMs before CCA is available. These works share similar design goals with Zion. Compared to them, Zion expands on isolation mode design, memory isolation and management, and shared memory, achieving higher security, flexibility, and efficiency, in addition to differences in underlying architecture.

VII. Conclusion

Zion is a practical confidential VM architecture built on commodity RISC-V processors. In addition to ensuring hardware compatibility, Zion relies on innovative designs in isolation mode, vCPU protection and update, memory isolation and management, and secure memory sharing to ensure flexibility, security, and efficiency. Evaluations show that Zion's performance overhead is less than 5% in most real-world applications.

REFERENCES

[1] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/Ispa*, vol. 1. IEEE, 2015, pp. 57–64.

[2] P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Trusted execution environments: properties, applications, and challenges," *IEEE Security & Privacy*, vol. 18, no. 2, pp. 56–60, 2020.

[3] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[4] A. Sev-Snp, "Strengthening vm isolation with integrity protection and more," *White Paper, January*, vol. 53, pp. 1450–1465, 2020.

[5] I. Corporation, "Intel Trust Domain Extensions (intel TDX) whitepaper," 2024, accessed: 2024-11-20. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf

[6] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, "Design and verification of the arm confidential compute architecture," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 465–484.

[7] A. W. Services, "Sev-snp support on amazon ec2," 2024, accessed: 2024-11-20. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html

[8] Microsoft, "Confidential virtual machine options on azure," 2024, accessed: 2024-11-20. [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-machine-options

[9] G. Cloud, "Confidential vm overview," 2024, accessed: 2024-11-20. [Online]. Available: https://cloud.google.com/confidential-computing/confidential-vm/docs/confidential-vm-overview

[10] R.-V. Foundation, "The RISC-V privileged architecture, version 1.10," 2017, accessed: 2024-11-20. [Online]. Available: https://riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf

[11] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 857–874.

[12] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[13] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Scalable memory protection in the penglai enclave," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 275–294.

[14] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "Cure: A security architecture with customizable and resilient enclaves," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1073–1090.

[15] J. Wang, P. Mahmoody, F. Brasser, P. Jauernig, A.-R. Sadeghi, D. Yu, D. Pan, and Y. Zhang, "Virtee: A full backward-compatible tee with native live migration and secure i/o," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 241–246.

[16] R. Sahita, V. Shanbhogue, A. Bresticker, A. Khare, A. Patra, S. Ortiz, D. Reid, and R. Kanwal, "Cove: Towards confidential computing on risc-v platforms," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, 2023, pp. 315–321.

[17] RISC-V Tech AP TEE, "Risc-v technical advisory panel (ap) tee mailing list," 2024, accessed: 2024-11-20. [Online]. Available: https://lists.riscv.org/g/tech-ap-tee

[18] RISC-V, "Risc-v specification status," 2024, accessed: 2024-11-20. [Online]. Available: https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154861/RISC-V+Specification+Status

[19] D. A. Patterson and D. R. Ditzel, "The case for the reduced instruction set computer," *ACM SIGARCH Computer Architecture News*, vol. 8, no. 6, pp. 25–33, 1980.

[20] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai, and R. A. Popa, "An off-chip attack on hardware enclaves via the memory bus," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[21] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[22] A. Cui and R. Housley, "Badfet: Defeating modern secure boot using second-order pulsed electromagnetic fault injection," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[23] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, "Voltpillager: Hardware-based fault injection attacks against intel sgx enclaves using the svid voltage scaling interface," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 699–716.

[24] R. Guanciale, H. Nemati, C. Baumann, and M. Dam, "Cache storage channels: Alias-driven attacks and verified countermeasures," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 38–55.

[25] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[26] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[27] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "Dawg: A defense against cache timing attacks in speculative execution processors," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 974–987.

[28] M. Orenbach, A. Baumann, and M. Silberstein, "Autarky: Closing controlled channels with self-paging enclaves," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[29] D. Li, Z. Mi, Y. Xia, B. Zang, H. Chen, and H. Guan, "Twinvisor: Hardware-isolated confidential virtual machines for arm," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 638–654.

[30] J. Yang, A. Cui, S. Stolfo, and S. Sethumadhavan, "Concurrency attacks," in *4th USENIX Workshop on Hot Topics in Parallelism (HotPar 12)*, 2012.

[31] R.-V. Foundation, "Iopmp specification," 2024, accessed: 2024-11-20. [Online]. Available: https://github.com/riscv-non-isa/iopmp-spec

[32] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 640–656.

[33] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, "Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v," in *Proceedings 2019-Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019.

[34] D. Schrammel, M. Waser, L. Lamster, M. Unterguggenberger, and S. Mangard, "Spear-v: Secure and practical enclave architecture for risc-v," in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023, pp. 457–468.

[35] X. Xu, W. Wang, Y. Wu, C. Wang, H. Zhu, H. Ma, Z. Min, Z. Pang, R. Hou, and Y. Jin, "virtcca: Virtualized arm confidential compute architecture with trustzone," *arXiv preprint arXiv:2306.11011*, 2023.

FF-INT8: Efficient Forward-Forward DNN Training on Edge Devices with INT8 Precision

Jingxiao Ma
School of Engineering
Brown University
Providence, RI
jingxiao_ma@alumni.brown.edu

Priyadarshini Panda
Department of Electrical Engineering
Yale University
New Haven, CT
priya.panda@yale.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI
sherief_reda@brown.edu

***Abstract*—Backpropagation has been the cornerstone of neural network training for decades, yet its inefficiencies in time and energy consumption limit its suitability for resource-constrained edge devices. While low-precision neural network quantization has been extensively researched to speed up model inference, its application in training has been less explored. Recently, the Forward-Forward (FF) algorithm has emerged as a promising alternative to backpropagation, replacing the backward pass with an additional forward pass. By avoiding the need to store intermediate activations for backpropagation, FF can reduce memory footprint, making it well-suited for embedded devices. This paper presents an INT8 quantized training approach that leverages FF's layer-by-layer strategy to stabilize gradient quantization. Furthermore, we propose a novel "look-ahead" scheme to address limitations of FF and improve model accuracy. Experiments conducted on NVIDIA Jetson Orin Nano board demonstrate 4.6% faster training, 8.3% energy savings, and 27.0% reduction in memory usage, while maintaining competitive accuracy compared to the state-of-the-art.**

***Index Terms*—Neural network quantization, Low power, Low-precision training, Resource-constrained devices.**

I. Introduction

Deep neural networks (DNNs) have achieved state-of-the-art performance across many application domains. However, their growing complexity significantly raises the computational cost of training, doubling roughly every six months from 2010 to 2022 [1]. For example, ResNet-50, which is introduced in 2015, consists of 26 million parameters and takes about 14 days to train on an NVIDIA M40 GPU [2]. In contrast, GPT-3, a large language model released in 2020, which contains 175 billion parameters, requires 3,640 PF-days to train, equivalent to 355 years of single-processor computing time, consuming 284,000 kWh of energy and costing more than 4.6 million US dollars [3]. These trends pose even greater challenges for edge devices, where limited memory and power resources make efficient, real-time DNN training an urgent priority.

The last few years have seen various methodologies for efficient deep learning computing, which can be categorized into pruning, quantization, neural architecture search, *etc* [4]. Among these, quantization stands out for its clear impact on memory footprint and computational cost, achieved by reducing the precision of weights and activations without altering the number of learned features [5]. Moreover, many edge devices are equipped with specialized hardware optimized for low-precision computations, making quantization particularly well-suited for edge computing scenarios. Despite its benefits, most quantization techniques are limited to model inference [6]. Meanwhile, the Forward-Forward (FF) algorithm [7] offers an alternative to backpropagation, replacing the backward pass with an additional forward pass to address inefficiencies in gradient computation. This paper delves into yet another advantage of the FF algorithm, specifically its applicability in implementing INT8 quantized training algorithms. The contributions of this paper are outlined below.

- To the best of our knowledge, FF-INT8 is the first work to devise a low-precision training method leveraging the Forward-Forward algorithm. Our observation reveals that FF algorithm's layer-wise greedy approach effectively mitigates the accumulation of accuracy degradation, a common issue in quantized backpropagation techniques.
- We propose a novel training method based on the Forward-Forward algorithm, named FF-INT8. This approach adopts a greedy, layer-wise training strategy, where each layer of a deep neural network is trained independently using INT8 precision, making it especially well-suited for resource-constrained edge devices.
- We identify two key drawbacks of the Forward-Forward algorithm: low convergence accuracy and slow convergence speed. To address these limitations, we propose a novel "look-ahead" scheme that incorporates information from subsequent layers. This enhancement improves both training accuracy and convergence efficiency, particularly in low-precision training environments.
- Using FF-INT8, we train multiple DNNs on the Jetson Orin Nano board, a typical edge device. Our experimental results show that FF-INT8 significantly improves efficiency while preserving high accuracy. We achieve a speedup of 4.6% in DNN training, reduce the memory footprint by 27.0% and energy consumption by 8.3% compared to a state-of-the-art INT8 training algorithm.

The organization of this paper is as follows. In section II, we overview relevant previous works in DNN quantization. We then briefly introduce Forward-Forward algorithm as background in section III. In section IV, we introduce FF-INT8 training algorithm. We provide our experimental results in section V. Finally, we summarize our conclusion in Section VI.

This work was supported in part by the National Science Foundation (Grant #2350180, #2453413, #2312366, and #2318152).

II. Previous Work

A wide range of quantization methods have been proposed to enhance the efficiency of DNN inference, with post-training quantization (PTQ) being the most common approach. PTQD [8] proposes a PTQ framework for diffusion models that optimizes model efficiency and performance by disentangling and correcting quantization noise. To make the model more robust to the reduced precision, methodologies of quantization-aware training (QAT) are proposed to preserve end-to-end model accuracy post quantization [9], [10].

Compared to the large amount of studies on accelerating inference by model quantization, few works explore low-precision training. MPTraining [11] trains DNNs using 16-bit floating-point. Minifloat [12] also proposes to use 8-bit floating-point numbers to train DNNs. Floating-point quantization requires specific hardware platform to achieve acceleration, which is not conducive to the practical deployment on resource-constrained devices. On the other hand, compared to floating-point, quantizing into 8-bit integer has great potential in hardware acceleration, since INT8 operations are widely supported by recent GPUs. UI8 [13] utilizes direction sensitive gradient clipping and deviation counteractive learning rate scaling to implement a fully unified INT8 training for convolutional neural networks. DAI8 [14] achieves better gradient quantization by performing channel-by-channel gradient distribution perception, but it sets too many channel dimension quantization parameters, adding additional computational complexity. GDAI8 [15] also uses a data-aware dynamic quantization scheme to quantize various special gradient distributions. In summary, most INT8 training methods require extra effort to adapt to gradient distribution.

III. Background

The traditional backpropagation (BP) algorithm is inefficient in several ways. First, it requires substantial memory to store the full computational graph, leading to a large memory footprint. Additionally, the backward pass of derivative computation takes much longer time and consumes more energy. In contrast, the Forward-Forward (FF) training algorithm [7] offers a novel direction to DNN training by replacing the forward and backward passes of BP with two forward passes to improve efficiency. Unlike BP, the FF algorithm uses two distinct data sets with opposing objective functions and trains the DNN layer-by-layer in a greedy manner. Figure 1 compares training procedure between BP and FF. In FF (Figure 1b), the first step is to generate "positive" and "negative" datasets from input data. One way to generate datasets is to annotate the input data using one-hot encoding vector of the label [7]. More specifically, the one-hot vectors in positive samples are pointing to the true labels, while the ones in negative samples are pointing to the wrong labels. FF algorithm uses *goodness* function G to qualify how well each layer is trained. The positive pass operates on positive samples and adjusts the weights to increase the goodness in every hidden layer, while the negative pass operates on negative samples and adjusts the weights to decrease the goodness. One common measurement

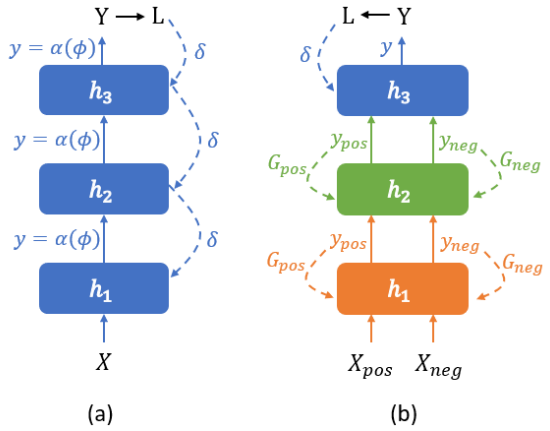


Fig. 1. (a) Backpropagation consists of a forward pass and a backward pass. (b) The Forward-Forward algorithm uses "positive" and "negative" datasets, and trains each layer individually using "goodness" function G.

of goodness function is the sum of squared neural activities. FF algorithm aims to make the goodness function well above a certain threshold θ for positive data (G_{pos}) and well below θ for negative data (G_{neg}), where θ can be considered as a hyperparameter that controls scale of the weights.

FF algorithm trains each layer once until convergence and then optimize its successor. The main idea behind such greedy scheme is to make each layer "excited" about positive samples and, at the same time, less excited about negative ones, so that positive samples can be trained to match the correct label later. FF algorithm improves efficiency in many aspects. First, instead of storing the entire computational graph, FF algorithm only stores current layer in memory, which significantly reduces memory footprint. Also, forward pass takes less time than backward pass. However, the heuristic property prevents earlier layers to learn from the later ones and final outputs. where training of earlier layers can be misleading. As a result, the FF algorithm often suffers from accuracy loss and struggles to scale effectively to modern, large-scale DNNs.

IV. Proposed Methodology

In this section, we describe our proposed methodology of INT8 Forward-Forward algorithm for efficient DNN training. Our FF-INT8 training method builds upon symmetric uniform quantization (SUQ), which is one of the most efficient quantization methods due to its hardware-friendly computation [6].

A. Network Depth and Gradient Quantization

As a preliminary experiment, we train ResNet-18 using CIFAR-10 dataset, with backpropagation and INT8 quantization including gradients. Figure 2 compares the changes of loss and accuracy per epoch between 32-bit floating-point (FP32) and INT8 training. The network is trained properly with FP32 precision. However, the loss of INT8 training increases dramatically as soon as we start training, while the accuracy drops to random level. We assume that INT8 training fails due to the accumulation of quantization error through backward propagation of derivatives. To further prove this hypothesis, we train three multilayer perceptrons (MLP) with different number of layers on MNIST dataset with different

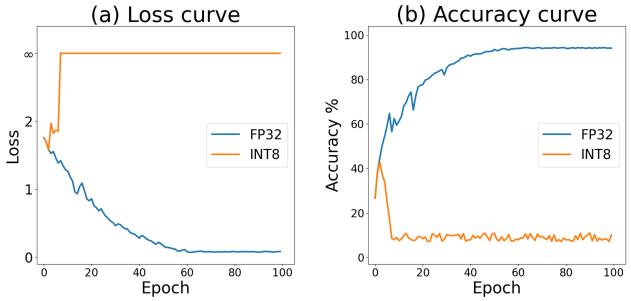


Fig. 2. Loss and accuracy of ResNet-18 on CIFAR-10 when gradients are directly quantized to INT8.

TABLE I
ACCURACY OF MULTILAYER PERCEPTRONS ON MNIST DATASET WITH DIFFERENT NUMBER OF HIDDEN LAYERS AND TRAINING PRECISION. EACH HIDDEN LAYER CONSISTS OF 500 NEURONS.

Number of Hidden Layers	FP32 Acc. (%)	INT8 Acc. (%)	Accuracy Difference (%)
0	89.5	88.7	-0.8
1	93.4	73.8	-19.6
2	94.5	62.4	-32.1
3	94.3	65.2	-29.1

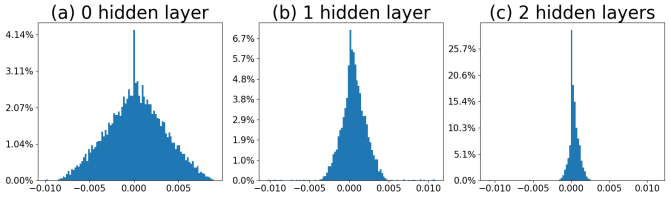


Fig. 3. Gradient distribution of first layer with different num. of hidden layers.

number of hidden layers, each of which is trained with FP32 and INT8 precision respectively. As Table I shows, as the number of hidden layers increases, the accuracy of FP32 training increases until the model overfits. On the contrary, the accuracy of INT8 training decreases dramatically as the network becomes deeper. The accuracy difference between FP32 and INT8 training is considerably small with a single-layer network, but increases significantly as we include the first hidden layer. We may conclude that quantization error accumulates as network becomes deeper. Instead of deeper networks, INT8 training can be directly executed on a single-layer network. Finally, we plot gradient distribution of the *first* layers for each MLP in Table I, which are trained using FP32. As Figure 3 shows, for deeper networks, gradient distribution of earlier layers are sharper with larger extreme values, while distribution of single-layer network is more even. Thus, direct quantization in deeper network leads to large quantization error, since most gradients gather in a small range and we cannot tell the differences in such small range after quantization. On the other hand, direct quantization in single-layer network is less error-prone. FF algorithm proposes to train each layer individually in a greedy manner, making it particularly compatible with INT8 training.

B. INT8 Forward-Forward Algorithm

FF-INT8 algorithm trains each layer individually in a greedy manner using both positive and negative datasets, with only one layer being trained at a time. For each dataset (positive or

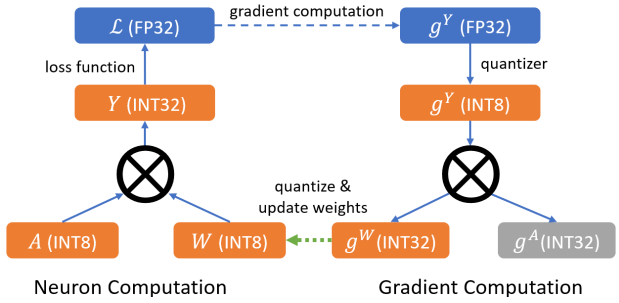


Fig. 4. Dataflow of FF-INT8 on a single layer.

negative), our FF-INT8 training workflow for a single layer is illustrated in Figure 4. Initially, we apply SUQ with stochastic rounding [16] to quantize the input data A. INT8 MAC operations are then employed to compute the dot product in either dense or convolution layers. Specifically, integer matrix multiplication with INT8 inputs and INT32 accumulation is used. The loss function is based on the negative log-likelihood of the goodness function G. For the positive dataset, the loss function $\mathcal{L}_{pos}$ is computed as follows:

$$\mathcal{L}_{pos} = -\log p(positive) = \log\left(1 + e^{-(G_{pos}-\theta)}\right) \quad (1)$$

For negative dataset, loss function $\mathcal{L}_{neg}$ is computed as

$$\mathcal{L}_{neg} = -\log p(negative) = \log\left(1 + e^{(G_{neg}-\theta)}\right) \quad (2)$$

where G represents the goodness function, and θ is the hyper-parameter that represents threshold as discussed in Section III.

By optimizing this loss function, we encourage outputs of positive samples to be large and outputs of negative samples to be small. The gradient g^Y is computed and quantized into INT8. Since gradients are not back-propagated to previous layers, there is no need to compute the gradients of input data g^A. Only gradients of weights g^W are computed and updated to current weights. Thus, for computation of neuron activity Y and gradient g_W, we replace floating-point computations with INT8, which saves a large amount of computation.

C. FF-INT8 Algorithm with "Look-Ahead"

As discussed in Section III, FF algorithm offers efficiency but imposes limitations by restricting each layer to learn solely from its immediate input, preventing communication with subsequent layers, particularly the final outputs. Consequently, earlier layers are primarily trained to distinguish between positive and negative data, leaving the task of specific label classification to the final layer. This lack of feedback from final outputs can lead to suboptimal training of earlier layers. As a result, FF algorithm often suffers from lower accuracy and requires more epochs to converge compared to backpropagation. These challenges are exacerbated in deeper networks, rendering the FF algorithm less effective for training large-scale DNNs. Additionally, FF algorithm struggles to accommodate widely-used structures such as residual blocks, which allow inputs to bypass certain layers and contribute directly to the block's output [17]. To optimize residual blocks effectively, all layers within the block must be trained interactively. However, due to the layer-by-layer nature of FF algorithm, layers within the same block are trained independently, without awareness of subsequent layers, leading to significant accuracy loss.

Algorithm 1: FF-INT8 with "Look-Ahead"

Input : Training set $(\mathbf{X}, \mathbf{Y})$, Number of epochs n
Output: Trained network M with k layers

1 Initialize neural network M
2 Initialize hyperparameter $\lambda = 0$ in loss function
3 Generate positive and negative sets from $(\mathbf{X}, \mathbf{Y})$
4 **for** $num_epoch = 1, 2, \ldots, n$ **do**
5 Execute forward pass using INT8 precision
6 Compute goodness function $G = ||\mathbf{y}||^2$ for each layer
7 **for** *current layer* $= 1, 2, \ldots, k$ **do**
8 Compute loss for current layer using G
9 Update weights using gradients in INT8 precision
10 **end**
11 Increase hyperparameters λ in loss function
12 **end**

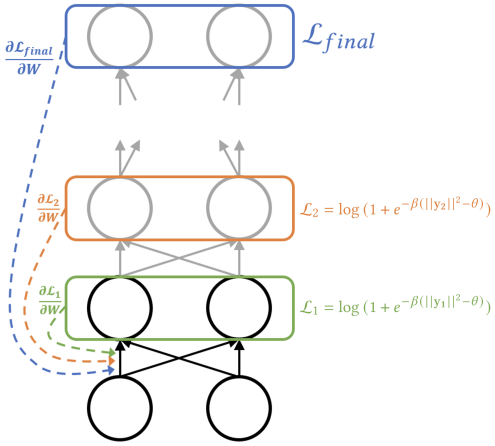


Fig. 5. Gradient computation of FF algorithm with "look-ahead", where loss functions of later layers are considered.

To solve these issues, our solution is to bridge connections between different layers, especially with final outputs, while retaining the efficiency of the FF algorithm. We name our method "*Look-Ahead*" scheme. In the standard FF algorithm, the loss function considers only the goodness function of the current layer during training. To enable feedback from subsequent layers, we redefine the loss function to incorporate the goodness functions of these layers. The revised loss function is defined as:

$$\mathcal{L}_{new} = \mathcal{L}_1 + \lambda \times (\mathcal{L}_2 + \mathcal{L}_3 + \cdots + \mathcal{L}_{final}) \tag{3}$$

Here, $\mathcal{L}_i$ represents the loss of the i^{th} layer, derived from the goodness function, and λ is a coefficient that balances the contributions of the current layer and the subsequent layers. Figure 5 illustrates the derivative computation with this modified loss function, as follow:

$$\frac{\partial \mathcal{L}_{new}}{\partial W} = \frac{\partial \mathcal{L}_1}{\partial W} + \lambda \times \frac{\partial (\mathcal{L}_2 + \mathcal{L}_3 + \cdots + \mathcal{L}_{final})}{\partial W} \tag{4}$$

In Section IV-B, we described a methodology where each layer is trained for n epochs independently, without considering information from later layers. For a neural network with k layers, this approach requires a total of $k \times n$ derivative computations. With the new loss function introduced in Equation 3, which involves derivatives from subsequent layers, we must perform complete forward passes for each derivative calculation. For the sake of efficiency, we propose Algorithm 1 to utilize *one* forward pass to update all layers. Specifically, for each epoch, we execute complete forward pass (line 5), and compute goodness function G for each layer (line 6). We can then construct loss functions of all layers as Equation 3 using goodness function of each layer (line 8), and compute the gradient (line 9). This approach facilitates interaction between layers, enabling them to optimize simultaneously and improve accuracy. Importantly, the goodness function for each layer depends solely on its neuron values, and the loss function is based solely on the goodness function. As a result, constructing the backward derivative chain is unnecessary, maintaining the computational efficiency of the loss and derivative calculations as vanilla FF-INT8 algorithm. Overall, the total number of derivative computations remains $k \times n$, the same as before. As later demonstrated in Section V-B, the "look-ahead" scheme significantly reduces the number of epochs needed for convergence, further improving efficiency. Regarding memory usage, since this scheme requires a full forward pass to compute the goodness function of later layers, all network weights must be retained in memory, resulting in a modest increase in memory footprint. However, compared to backpropagation, this approach still saves memory by avoiding the storage of intermediate activations and the full gradient chain, as it does not rely on a complete backward pass.

One more hyperparameter in "FF-INT8 Algorithm with look-ahead" is λ, which balances between current layer and other layers. For first few epochs, since each layer is less optimized, our priority is to train each layer so that it can basically tell between positive samples and negative samples. In other word, we want to discourage the interaction between layers at the beginning of training, since later layers are not optimized and information of later layers does not help the optimization of current layer. After a few epochs, each layer is better optimized. We then gradually increase λ to encourage interaction between layers to improve convergence accuracy.

V. Experimental Results

In this section, we first introduce our experimental setup. The first experiment is to compare FF-INT8 algorithm with and without "look-ahead", to show the benefit of it. We then theoretically analyze and compare the number of each operation, demonstrating that our methodology reduces computational cost. Finally, we comprehensively use our methodology to train different DNNs, and compare against other methodologies to demonstrate state-of-the-art performance.

A. Experimental Setup

1) Datasets and models: As shown in Table II, we evaluate four different DNNs as benchmarks. Since the experiments are conducted on an edge device with limited computation resources, we mainly use CIFAR10 as training dataset. The batch size is set to 32.
2) Hardware setup: The NVIDIA Jetson Orin Nano board, equipped with an INT8 engine, is used to measure training time, energy and memory usage. The device specifications, detailed in Table III, demonstrate that it is representative of edge devices used for DNN training.

TABLE II
DNN ARCHITECTURES AND DATASETS

DNN	Dataset	Num. of Params (M)
Multi-layer perceptron (MLP) (2 hidden layers)	MNIST	1.79
MobileNet-V2 [18]	CIFAR10	2.24
EfficientNet-B0 [19]		3.39
ResNet-18 [17]		11.19

TABLE III
TECHNICAL SPECIFICATIONS OF NVIDIA JETSON ORIN NANO

GPU	512-core NVIDIA Ampere architecture GPU
CPU	6-core Arm®Cortex®-A78AE v8.2 64-bit CPU
Memory	4GB 64-bit LPDDR5 34 GB/s
Power	7W-10W
AI Performance	20 TOPS

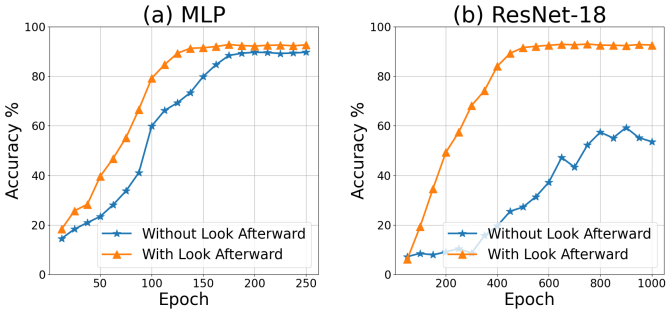


Fig. 6. Test accuracy across different epochs for MLP and ResNet-18 trained using FF-INT8, with and without "look-ahead" scheme respectively.

3) Hyperparameters: In Equation 1 and 2, we set threshold $\theta = 2.0$. In Equation 3, λ is initialized to 0, and increased by 0.001 each epoch.

B. Training with "Look-Ahead"

In the first set of experiment, we show that "look-ahead" scheme improves accuracy for FF-INT8 as described in Section IV-C. We first train MLP with 2 hidden layers using both FF-INT8 algorithms and demonstrate results in Figure 6(a). Without "look-ahead" scheme, the accuracy converges to nearly 90% with 180 epochs. However, with "look-ahead" scheme, MLP reaches slightly higher convergence accuracy with only 130 epochs. Thus, we argue that by interaction with later layers, earlier layers compute gradients and update weights in a "more optimized way", which leads to higher convergence accuracy and faster training.

Figure 6(b) demonstrates the results in ResNet-18, which consists of residue blocks. Different from MLP, if ResNet-18 is trained without "look-ahead" scheme, the accuracy converges to only 60%, while the accuracy curve is very unstable. There are two reasons for the bad performance:

1) The scale of ResNet-18 is much larger compared to MLP, which means it is much difficult for a greedy approach like FF algorithm to find the optimized solution.
2) Residue block allows input to be skipped over few layers and accumulated to block's output. However, without "look-ahead" scheme, when previous layers in a block are trained, the vanilla FF algorithm does not consider the accumulation operation in block's output layer. And when block's output layer is trained, the previous layers have already trained and cannot be modified. Thus, the optimization process is limited, which leads to low convergence accuracy.

TABLE IV
COMPARISON OF COMPUTATIONAL COST BETWEEN INT8 FORWARD-FORWARD ALGORITHM AND INT8/FP32 BACKPROPAGATION

Computation		Operation	Counts (OPs)
FF-INT8	Quantization Phase	32-bit CMP	32.4K
		32-bit FADD	165.9K
	MAC Phase	8-bit MUL	23.8M
		8-bit ADD	23.8M
BP-FP32	MAC Phase	32-bit FADD	898.2M
		32-bit FMUL	898.2M
GDAI8 [15] (BP-INT8)	Quantization Phase	32-bit CMP	7.2K
		32-bit FADD	18.4K
	MAC Phase	8-bit MUL	898.2M
		8-bit ADD	898.2M

As Figure 6(b) shows, by considering subsequent layers, "look-ahead" scheme solves both problems and significantly improves convergence accuracy. Given that many modern DNN architectures incorporate residual blocks, such method significantly enhances the FF-INT8 algorithm.

C. Analysis of Computational Cost

In Section V-B, our experiment suggests that FF-INT8 training requires a large number of epochs to converge. In this section, we analyze the theoretical computational cost to demonstrate that although FF-INT8 training has much more epochs compared to backpropagation, it is still efficient. We count the number of required operations to train 4-layer MLP using MNIST dataset with three settings, *i.e.* the proposed FF-INT8 training method with "look-ahead", and backpropagation with 32-bit floating-point (BP-FP32) as baseline. We also include Gradient Distribution-Aware INT8 training algorithm (GDAI8) [15] for comparison, which is an INT8 training algorithm based on backpropagation. Table IV summarize the amount of computation required for training a mini-batch of 10 samples for three approaches. In FF-INT8, we have a quantization phase and a multiply–accumulation (MAC) phase, where computation of quantization phase is negligible compared to MAC. Comparing MAC phase, FF-INT8 training requires 23.8M 8-bit MAC operations, whereas backpropagation approach (BP-FP32 or GDAI8) requires 898.2M MAC operations (FP32 or INT8). This is because the FF algorithm does not have large matrix multiplication to back-propagate derivatives from the last layer to the first. Thus, per mini-batch, FF-INT8 only requires 2.6% of MAC operations in backpropagation approach. INT8 arithmetic is also 4x faster than FP32 in hardware, which makes FF-INT8 hundres of times more efficient compared to BP-FP32 per epoch. Additionally, FF-INT8 only computes forward pass and does not compute backward pass of gradients. In many devices, forward pass is more efficient due to hardware optimization for model inference. In conclusion, although FF-INT8 training needs a large number of epochs, theoretically it is still more efficient compared to both backpropagation approaches.

TABLE V

SUMMARY OF MODEL ACCURACY, TRAINING TIME, ENERGY CONSUMPTION AND MEMORY FOOTPRINT BETWEEN DIFFERENT APPROACHES. TRAINING ALGORITHMS ARE BASED ON EITHER BP (BACKPROPAGATION) OR FF (THE FORWARD-FORWARD ALGORITHM). THE SUFFIX DENOTES THE PRECISION, WHERE FP32 MEANS 32-BIT FLOATING-POINT, AND INT8 MEANS QUANTIZING TO 8-BIT INTEGER. UI8 [13] REFERS TO UNIFIED INT8 TRAINING ALGORITHM, AND GDAI8 [15] REFERS TO GRADIENT DISTRIBUTION-AWARE INT8 TRAINING ALGORITHM.

Model	Training Algorithm	Accuracy (%)	Time (s)	Energy (J)	Memory (MB)
MLP	BP-FP32	94.5	482.3	2315.0	247.6
	BP-INT8	52.4	326.1	1206.6	213.9
	BP-UI8 [13]	92.3	335.2	1277.1	197.0
	BP-GDAI8 [15]	93.8	344.9	1345.4	182.6
	FF-INT8	94.3 (-0.2) (+0.5)	312.7 (-35.2%) (-9.3%)	1097.0 (-52.6%) (-18.4%)	140.7(-43.2%) (-22.9%)
MobileNet-v2	BP-FP32	91.5	2370.8	11593.2	649.8
	BP-INT8	5.9	1851.6	7836.0	571.6
	BP-UI8 [13]	87.2	1960.0	7618.5	592.6
	BP-GDAI8 [15]	90.9	1790.7	6528.1	578.9
	FF-INT8	91.1 (-0.4) (+0.2)	1703.9 (-28.1%) (-4.8%)	6174.3 (-46.7%) (-5.4%)	437.0 (-32.7%) (-24.5%)
EfficientNet-B0	BP-FP32	89.4	2692.8	13356.2	861.0
	BP-INT8	11.8	2095.0	8563.9	703.9
	BP-UI8 [13]	85.3	2230.8	8656.2	735.5
	BP-GDAI8 [15]	88.9	2177.1	8589.9	692.0
	FF-INT8	88.6 (-0.8) (-0.2)	2129.9 (-20.9%) (-2.2%)	8093.8 (-39.4%) (-5.8%)	505.2 (-41.3%) (-27.0%)
ResNet-18	BP-FP32	93.5	3853.0	18764.1	1096.4
	BP-INT8	7.2	2676.1	10436.8	885.8
	BP-UI8 [13]	89.7	2873.8	11466.5	920.7
	BP-GDAI8 [15]	92.9	2751.6	10291.0	894.1
	FF-INT8	93.1 (-0.4) (+0.2)	2697.9 (-30.0%) (-2.0%)	9926.5 (-47.1%) (-3.5%)	682.3 (-37.7%) (-23.7%)
Avg. difference between FF-INT8 and BP-FP32 (baseline)		**Reduce 0.4%**	**Save 28.6%**	**Save 46.4%**	**Save 38.7%**
Avg. difference between FF-INT8 and BP-GDAI8 (state-of-the-art)		**Improve 0.2%**	**Save 4.6%**	**Save 8.3%**	**Save 27.0%**

D. Accuracy, Time, Energy, Memory Footprint

For the last set of experiment, we compare model accuracy, training time, energy consumption between multiple training algorithms, and demonstrate results in Table V. As *baseline*, we train DNNs using backpropagation (BP) with 32-bit floating-point operations, denoted as BP-FP32. If directly quantizing gradients to INT8 using BP (BP-INT8), although we observe significant time, energy and memory savings, accuracy decreases dramatically due to the accumulation of quantization error, as analyzed in Section IV-A. With deeper architecture, direct quantization makes training even worse.

As a comparison, we train each DNN with two existing INT8 training algorithms, *i.e.* Unified INT8 Training Algorithm (BP-UI8) [13] and Gradient Distribution-Aware INT8 Training Algorithm (BP-GDAI8) [15]. Both algorithms are based on BP, and quantize gradients based on analysis of gradient distribution. Since they use BP with additional operation of gradient monitoring and analysis, the computational cost of both is slightly higher than direct quantization (BP-INT8). The training accuracy is close to traditional BP with FP32 operands (baseline). BP-GDAI8 has the *state-of-the-art* performance in terms of trade-off between model accuracy and efficiency, which is chosen as the comparison model.

Finally, we implement our FF-INT8 with "look-ahead" as described in Section IV-C. As analysis in Section IV-B, gradient quantization works better on FF algorithm due to its layer-by-layer greedy manner. Thus, the training accuracy is much higher compared to direct quantization in BP (BP-INT8). By comparing to BP-GDAI8, we show that training accuracy of FF-INT8 is 0.2% higher against the state-of-the-art, and is very close to the traditional backpropation (BP-FP32), with a 0.4% reduction. For training time and energy consumption, since Jetson Orin Nano board has an INT8 engine, we observe a significant improvement compared to training with FP32 operations. Our FF-INT8 algorithm is also more efficient compared to BP-GDAI8, with 4.6% saving in training time and 8.3% saving in energy consumption.

As mentioned in Section III, one of the major benefits of FF algorithm is smaller memory footprint. Normally, BP algorithms rely on automatic differentiation, which requires to store the large computational graph for gradient backpropagation in memory. But since FF algorithm does not have backward pass, it does not need to store this computational graph. Such improvement in memory footprint is further enhanced by INT8 operations. Thus, compared to BP-GDAI8, FF-INT8 saves memory footprint by 27.0%.

VI. CONCLUSION

In this paper, we present FF-INT8, a novel low-precision training methodology that leverages the Forward-Forward algorithm for INT8 quantized training. Unlike backpropagation-based approaches, FF-INT8 stabilizes gradient quantization by employing a layer-wise greedy training strategy, effectively mitigating accuracy degradation. To further enhance convergence and accuracy, we propose the "look-ahead" scheme, which enables earlier layers to incorporate feedback from subsequent layers, addressing a key limitation of the standard FF algorithm. By comparing against the state-of-the-art approach, we demonstrate that FF-INT8 significantly reduces training time, energy cost and memory footprint while maintaining high accuracy. FF-INT8 offers a promising alternative for energy-efficient neural network training.

REFERENCES

[1] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, "Compute trends across three eras of machine learning," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[2] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes," in *Proceedings of the 47th international conference on parallel processing*, 2018, pp. 1–10.

[3] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[4] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on tinyml," *IEEE Access*, 2023.

[5] J. Ma and S. Reda, "Wenet: Configurable neural network with dynamic weight-enabling for efficient inference," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2023, pp. 1–6.

[6] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[7] G. Hinton, "The forward-forward algorithm: Some preliminary investigations," *arXiv preprint arXiv:2212.13345*, 2022.

[8] Y. He, L. Liu, J. Liu, W. Wu, H. Zhou, and B. Zhuang, "Ptqd: Accurate post-training quantization for diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[10] X. Chu, L. Li, and B. Zhang, "Make repvgg greater again: A quantization-aware approach," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 10, 2024, pp. 11 624–11 632.

[11] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[12] S. Fox, S. Rasoulinezhad, J. Faraone, P. Leong *et al.*, "A block minifloat representation for training deep neural networks," in *International Conference on Learning Representations*, 2020.

[13] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1969–1979.

[14] K. Zhao, S. Huang, P. Pan, Y. Li, Y. Zhang, Z. Gu, and Y. Xu, "Distribution adaptive int8 quantization for training cnns," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 4, 2021, pp. 3483–3491.

[15] S. Wang and Y. Kang, "Gradient distribution-aware int8 training for neural networks," *Neurocomputing*, vol. 541, p. 126269, 2023.

[16] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[19] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

Versatile Cross-platform Compilation Toolchain for Schrödinger-style Quantum Circuit Simulation

Yuncheng Lu, Shuang Liang, Hongxiang Fan, Ce Guo, Wayne Luk, Paul H. J. Kelly
Department of Computing, Imperial College London, London, UK
Email: {yuncheng.lu19, shuang.liang, hongxiang.fan, c.guo, w.luk, p.kelly}@imperial.ac.uk

***Abstract*—While existing quantum hardware resources have limited availability and reliability, there is a growing demand for exploring and verifying quantum algorithms. Efficient classical simulators for high-performance quantum simulation are critical to meeting this demand. However, due to the vastly varied characteristics of classical hardware, implementing hardware-specific optimizations for different hardware platforms is challenging. To address such needs, we propose CAST (Cross-platform Adaptive Schrödinger-style Simulation Toolchain), a novel compilation toolchain with cross-platform (CPU and Nvidia GPU) optimization and high-performance backend supports. CAST exploits a novel sparsity-aware gate fusion algorithm that automatically selects the best fusion strategy and backend configuration for targeted hardware platforms. CAST also aims to offer versatile and high-performance backend for different hardware platforms. To this end, CAST provides an LLVM IR-based vectorization optimization for various CPU architectures and instruction sets, and a PTX-based code generator for Nvidia GPU support. We benchmark CAST against IBM Qiskit, Google QSimCirq, Nvidia cuQuantum backend, and other high-performance simulators. On various 32-qubit CPU-based benchmarks, CAST achieves up to 8.03x speedup than Qiskit. On various 30-qubit GPU-based benchmarks, CAST achieves up to 39.3x speedup than Nvidia cuQuantum backend.**

I. Introduction

Significant research efforts have focused on developing new quantum algorithms under the assumption of reliable quantum computation [1], [2]. Nevertheless, verifying these algorithms and demonstrating their performance at scale is challenging. Therefore, there is a growing demand for simulators based on classical computers as alternatives to quantum hardware. High-performance and reliable classical simulators play a vital role in identifying the potential of quantum computers, particularly in scenarios where quantum computers claim supremacy in tackling problems that are intractable for classical systems [3], [4], [5], [6]. The performance of simulations also becomes crucial for various applications. For instance, Variational Quantum Algorithms (VQA) [7] often require optimizing parametrized quantum circuits by simulating thousands of runs on the same circuit structure. Similarly, Hamiltonian Evolution Simulation relies on the decomposition of the time-evolution operator [8], which induces a significant circuit depth.

However, simulating the quantum world using classical computers has been recognized as a challenging problem. To perfectly mimic the physics of the quantum world, it requires either exponentially increasing memory or time with respect to system size [9]. Among current simulation approaches, Schrödinger style simulations are often used to facilitate the comprehensive verification and evaluation of quantum algorithms. These methods require exponential memory to keep track of all quantum state information and are capable of handling all kinds of quantum operations.

Although various Schrödinger style simulators have been introduced by both industry [10], [11], [12] and academia [13], [14], [15], they are suffering two key issues in obtaining accessible and high-performant simulations. First, the absence of advanced and versatile simulation-oriented gate rewriting and fusion optimization hampers their scalability for large quantum circuits. Second, existing simulation tools do not offer comprehensive cross-platform backend support, and there is an urgent need for an easy-to-use cross-platform solution to accommodate the diverse needs of users with varying hardware resources. To address these challenges, we propose an adaptive and cross-platform (with CPU and Nvidia GPU support) compilation toolchain for high-performance and scalable quantum circuit simulation named `CAST` (Cross-platform Adaptive Schrödinger-style Simulation Toolchain). `CAST` introduces a custom intermediate layer and a cost-model based adaptive gate fusion algorithm. The algorithm automatically selects the best fusion configuration according to the characteristics of targeted CPU and GPU platforms. `CAST` incorporates high-performance kernel generators that automate backend support by emitting vectorized LLVM IR codes for a versatile SIMD solution on CPU platforms and PTX codes for utilizing CUDA cores.

Overall, this work makes the following contributions:

- **A novel adaptive gate fusion technique:** This technique considers the sparsity patterns and categories with an agglomerative scheme to apply gate fusion. We also adopt a cost-model based optimization that automatically selects fusion configuration based on targeted hardware. As a result, our tool can apply CPU or GPU-specialized gate fusion optimization to best match the characteristics of the target hardware.
- **A dynamic kernel generation approach:** `CAST` enables customized optimization to reduce the number of operations needed to perform quantum circuit simulation. Together with the sparsity-aware fusion technique, `CAST` achieves fast and energy-efficient quantum circuit simulation.
- **A cross-platform quantum simulation framework:** `CAST` serves as a cross-platform automated compilation toolchain for high-performance quantum simulation on CPU and Nvidia GPU platforms, featuring adaptive front-end optimization algorithms and high-performance backend support.

II. Background and Motivation

Statevector-based Simulation Statevector-based simulation keeps track of all *amplitudes* in the statevector. An amplitude is a complex number, and an n-qubit statevector is a length-2^n complex vector with norm 1. Let ψ be the length-2^n complex vector that represents an n-qubit statevector. Applying a single-qubit gate U on qubit k is equivalently performing the matrix-vector multiplication specified by

$$\left(\mathsf{I}^{\otimes(n-k-1)} \otimes \mathsf{U} \otimes \mathsf{I}^{\otimes k}\right)\psi. \tag{1}$$

This expression represents multiplying a $2^n \times 2^n$ matrix with a $2^n \times 1$ vector. As the matrix is highly sparse, we can equivalently perform

TABLE I
COMPARISON OF VARIOUS CLASSICAL QUANTUM CIRCUIT SIMULATORS.

	Precision	CPU Backend SIMD Support[a]	CUDA Support	Gate Fusion			Dynamic Kernel
				Size	Sparsity	Parametrized	
QuEST [13]	`f32` and `f64`	✗	Builtin	✗	✗	✗	✗
Qibo [14]	`f32` and `f64`	External	CuPy	✓	Partial[b]	Limited	✗
Qulacs [15]	`f32` and `f64`	`f64` AVX2 only	Builtin	✓	✗	✓	✗
Qiskit [10]	`f32` and `f64`	`f64` AVX2 only	cuQuantum	✓	Partial[c]	✗	✗
QPanda [16]	`f64` only	`f64` AVX2 only	cuQuantum	✓	✗	Limited	✗
Cirq [11][d]	`f32` and `f64`	External	✗	✗	✗	✗	✗
QSimCirq [11]	`f32` only	`f32` AVX2 and `f32` AVX512	cuQuantum	Up to 6-qubit	✗	Limited	✗
CUDA Quantum [17]	`f32` and `f64`	External	cuQuantum	Up to 6-qubit	✗	Limited	✗
CAST (ours)	`f32` and `f64`	✓[e]	Builtin	✓	✓	✓	✓

[a]"External" means this backend depends on external pre-compiled library, such as numpy's C++ backend.
[b]Qibo provides `qibojit` as a just-in-time engine.
[c]Qiskit's gate fusion algorithm does not consider general gate matrix sparsity, but it separates diagonal gates and general unitary gates.
[d]Cirq and QSimCirq (sometimes QSim) are both developed by Google. While Cirq has its own simulation backend, QSimCirq provides a high-performance backend specifically optimized for SSE, AVX2, and AVX512-capable hardware for simulating Cirq circuits.
[e]Tested to support NEON, SSE, AVX2, and AVX512, both `f32` and `f64` precision.

2^{n-1} matrix-vector multiplications in sequential, each time between U and two of the amplitudes of ψ to improve the efficiency. That is,

$$\mathsf{U}\begin{bmatrix}\psi_{\alpha(t)}\\ \psi_{\beta(t)}\end{bmatrix}, \quad t=0,\cdots,2^{n-1}-1. \tag{2}$$

More generally, if U is a k-qubit gate represented by a $2^k\times 2^k$ unitary matrix, then applying U on an n-qubit statevector ψ consists of 2^{n-k} matrix-vector multiplications

$$\mathsf{U}\begin{bmatrix}\psi_{\alpha_0}(t)\\ \vdots\\ \psi_{\alpha_{2^k-1}}(t)\end{bmatrix}, \quad t=0,\cdots,2^{n-k}-1. \tag{3}$$

Ahead-of-time Kernel Preparation While a dense single-qubit gate can have up to 8 non-zero scalars, each Pauli gate only has two. Simulators with support to only general quantum gates inevitably lose optimizations for Pauli gates. Therefore, for better performance, a traditional approach is to hand-code popular gate kernels ahead of time. Some simulators such as QSimCirq even write vector intrinsics to gain more performance on selected CPU platforms.

However, such a kernel preparation scheme induces extreme workload in writing different kernels to support every gate. With gate fusion, the workload is even greater. Fused gates usually act on more than two qubits, so their sparsity patterns are much more diverse and unpredictable. The pattern of loading and storing vector amplitudes also grows exponentially with respect to the gate size. To reduce workload, some simulators supporting gate fusion, such as Qulacs, QPanda, and QSimCirq, invoke general multi-qubit kernel on fused gates. Some other ones, such as Qiskit, separate diagonal matrices by providing additional diagonal multi-qubit gate kernels. However, none of them could fully appreciate the sparsity patterns of fused gates.

Motivation To facilitate the quantum simulation on classical computers, various quantum circuit simulators have been developed by both industry and academia [13], [14], [15], [10], [16], [11]. However, as summarized in Table I, most of these tools overlook the sparsity information during simulation and have limited backend support. As illustrated in Fig. 1, to leverage the sparsity and dynamic optimization opportunities, our simulator proposes a novel sparsity-aware gate fusion technique and a dynamic kernel generation process

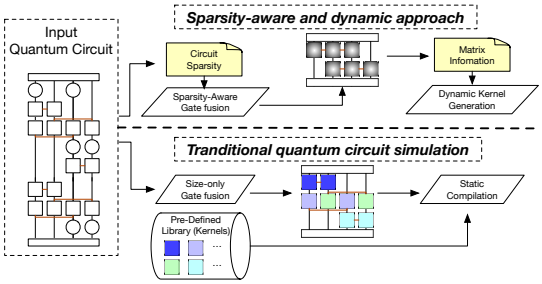


Fig. 1. The proposed CAST approach exploits sparsity information and runtime-matrix information to improve simulation performance.

that utilizes runtime matrix information for further optimizations. Compared with prior quantum circuit simulators, our tool enables more effective gate fusion with highly optimized kernels, significantly improving the simulation performance. By integrating these optimizations and backend solutions, our proposed toolchain delivers cross-platform, scalable, and high-performance quantum simulations.

III. OVERVIEW

A. CAST Compilation

`CAST` compilation consists of two main phases: an adaptive gate fuser and partitioner that transpiles input circuits via a custom data structure `CircuitTile`, and a dynamic kernel generator that emits and optionally compiles hardware-specific instructions. A schematic diagram is given in Fig. 2.

`CAST`'s workflow brings the following advantages:

- When targeting CPU based platforms, `CAST` provides versatile SIMD capabilities via LLVM IR backend. This avoids the traditional ahead-of-time approach by hard-coding multiple vector intrinsics for different SIMD instruction sets and simulation precisions.
- `CAST` optimizes sparse gates during kernel generation. For example, all zero entries in the matrix are skipped in generated kernels. This brings a cross-platform solution for the previously infeasible task of preparing optimized kernels for fused gates.
- `CAST` kernel generator is highly customizable and extendable. Kernel generators allow designers to obtain a consistent performance across platforms without repetitive work of implementing

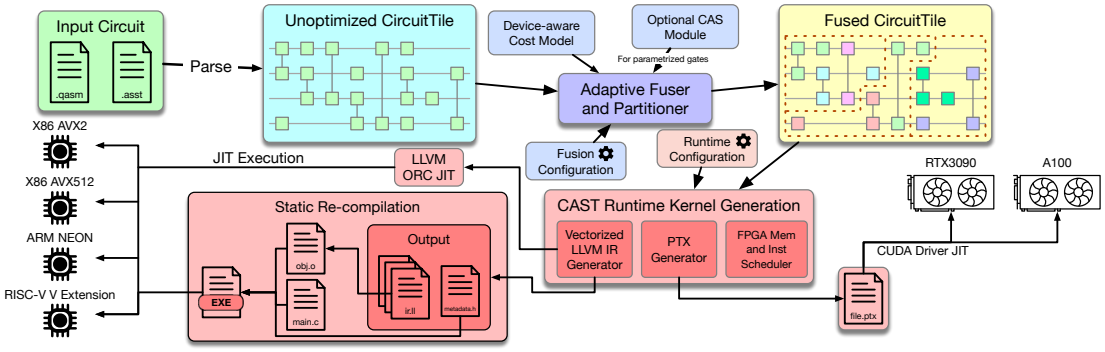


Fig. 2. CAST Workflow.

hardware-specific optimizations. Along the automation process, CAST also provides a lot of room for exploration as shown in Table II.

TABLE II
PART OF CAST SOFTWARE ARGUMENTS.

Argument	Command line
Precision	`-f32, -f64`
CPU vector size	`-S=<int>, -neon, -sse -avx2, -avx512`
Gate fusion	`-cost-model=<cost-model>, -fusion=<int> -zero-tolerance=<float> -one-tolerance=<float> -multi-traversal=<bool> -agglomerative=<bool>`
Kernel configurations	`-arch=<cpu/gpu> -use-fma, -use-fms, -use-pdep -alt-format, -sep-format -zero-tolerance-kernel=<float> -elem-sharing-threshold=<float> -use-imm-value=<bool> -use-const-mem-space=<bool> -force-dense-kernel=<bool>`

IV. ADAPTIVE SPARSITY-AWARE AGGLOMERATIVE GATE FUSION

A. `CircuitTile` Data Structure

Because the difficulty of gate fusion lies in preserving the commutation relation of quantum circuits, we design a custom data structure `CircuitTile` as an intermediate layer to represent quantum circuits and to perform gate fusion.

Internally, `CircuitTile` contains a linked list of arrays that we call *tile*. Each array in the tile is called a *row*. Every row is an array of null-able `GateBlock` pointers. A `GateBlock` is an aggregate of gates. Each gate keeps track of its left and right neighbors.

To add a gate to the graph, we create a `GateBlock` consisting of solely this gate and append it to the tile. Tile spans vertically by convention, so to append a `GateBlock` to the tile, we insert it to the uppermost vacant row.

B. Agglomerative Gate Fusion

Gates with disjoint target qubit sets always commute with each other. So in the tile, blocks on the same row can be safely fused together. On the other hand, gates not on the same row can be fused together only when they are connected and on consecutive rows.

In other words, a pair of gate blocks can be fused in two scenarios. The first scenario is **Commuting Gate Fusion**, where the two blocks are on the same row. The second scenario is **Consecutive Gate Fusion**, where the two blocks are located in consecutive rows, and there is at least one direct connection spanning between them.

CAST employs tile-based iteration to perform gate fusion. The pseudo-code is provided in Algorithm 1. This algorithm takes in a CircuitTile T and an unsigned integer k as the input. It traverses through all gate blocks and fuse the blocks. During the execution, the number of qubits of the fused gate should stay below k to avoid fragments. The algorithm returns **true** if and only if there is at least one pair of gate block merged during the execution.

Algorithm 1 Tile Traversal in Agglomerative Gate Fusion

```
1:  function TRAVERSE(CircuitTile T, unsigned k)
2:      δ ← false
3:      for r ∈ [0..N_rows − 1] do
4:          for q ∈ [0..N_qubits − 1] do
5:              b⊤ ← T_{r,q}
6:              if b⊤ ≠ ∅ then
7:                  b⊥ ← T_{r+1,q}
8:                  if b⊥ ≠ ∅ then
9:                      MOVEBLOCK(b⊤, b⊥)
10:                 if FUSIBLE(b⊤, b⊥, k) then
11:                     FUSE(b⊤, b⊥)              ▷ Consecutive Fusion
12:                     δ ← true
13:         for q ∈ [1..N_qubits − 1] do
14:             b⊤ ← T_{r,q−1}
15:             b⊥ ← T_{r,q}
16:             if FUSIBLE(b⊤, b⊥, k) then
17:                 FUSE(b⊤, b⊥)                  ▷ Commuting Fusion
18:                 δ ← true
19:     return δ
```

The algorithm actively relocates blocks to create fusion opportunities. Specifically, when the traversal reaches a gate block in the i-th row and the space below it in the $(i+1)$-th row is unoccupied, the algorithm moves the gate block down to the $(i+1)$-th row to fill the space. This relocation allows the block to potentially fuse with a block in the $(i+2)$-th row. For example, Fig. 3 gives an illustration of such a case. In the compact tile above, blocks 1 and 3 are not in consecutive rows. Since the position immediately below block 1 is empty (line 8 in Algorithm 1), block 1 can be moved down. Block 1 will then fuse with block 3 when we traverse to the new position of block 1.

To place fused blocks back into the tile, the FUSE procedure first removes $b_\top$ and $b_\perp$. If the removal creates enough space in rows r or $(r+1)$ to accommodate the fused block, the fused gate block occupies the corresponding space (prioritize on row $(r+1)$). Otherwise, the FUSE procedure inserts a new row between the two rows and places the fused block in the new row. Note that the creation of the new

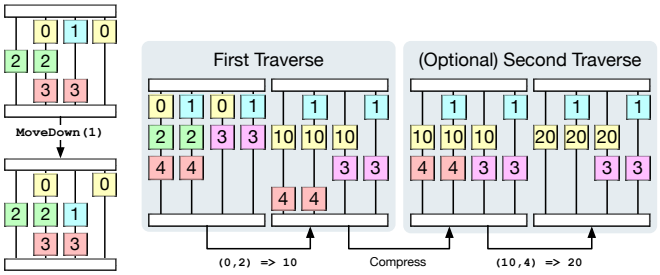


Fig. 3. `MoveDown`.

Fig. 4. Tile traversal.

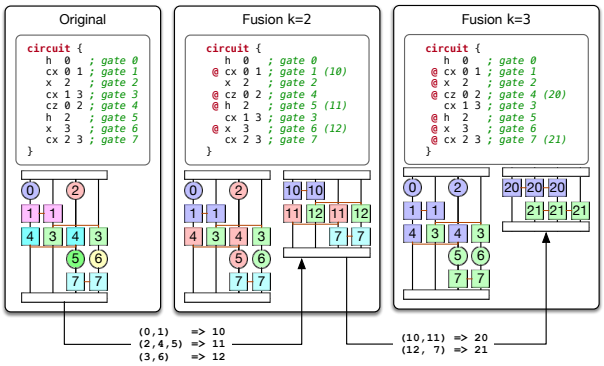


Fig. 5. Demonstration of agglomerative gate fusion. Each top block shows `CAST` serialization of the `CircuitTile`. The `@` symbol denotes fusing gates together.

row can leave empty spaces (Fig. 4). To maintain the compactness of the `CircuitTile`, we compress it at the end of each tile traversal.

Given a specified maximum size $k_{\max}$ of fused gates, the agglomerative scheme iteratively increases the allowed size of fused gates from 2 to $k_{\max}$. We found that such an incrementing scheme helps reduce fragments (small gates remaining after the algorithm finishes) and leaves with overall sparser gates.

C. Adaptive Fusibility Check

Adaptive gate fusion traverses the tile the same way as before, but adds extra criteria in fusion eligibility checks. For CPU and GPU based adaptive gate fusion, we define a cost model upon the total operation count and the number of blocks present in the fused circuit.

The *operation count* captures the sparsity of gates, and is defined as the number of run-time operations needed to perform matrix-vector multiplication. Take single-qubit gates as an example, the matrix-vector multiplication (2) corresponds to updating four variables (registers) in each iteration. For machines with support of both Fused Multiplication Addition (FMA) and Fused Multiplication Subtraction (FMS), every non-zero entry in the matrix requires 2 operations in run-time. A dense k-qubit gate has 2^{2k+1} entries, and thus its operation count is 2^{2k+2}. On the other hand, the operation count for sparse gates such as Pauli gates is far less than the worst case described above.

Fig. 6 shows one example. For a fixed number of threads, there is a peak performance point when we vary gate fusion aggressiveness. Left to the peak point is usually when memory bottlenecks speed, and right to the peak point is usually when computation power bottlenecks speed.

`CAST` provides several preset fusion configurations for CPU and GPU based platforms. Users are also encouraged to use the our provided `CostModel` class to conduct benchmarks for a cost model

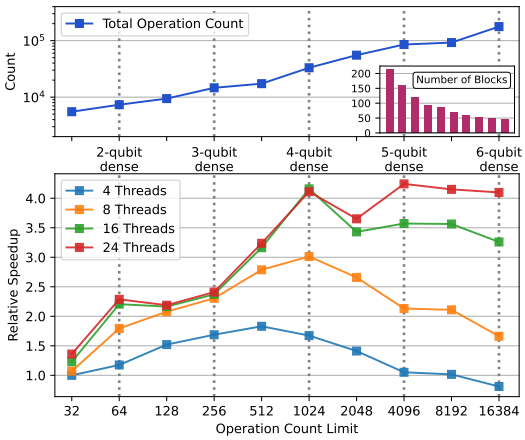


Fig. 6. Operation Count versus Simulation Time. Reference time is measured at the 4-thread 32-op-count-limit experiment. All experiments were carried out on CPU-based 30-qubit statevectors with double precision.

specialized to their hardware platform. `CostModel` will measure and record the running time of several quantum gates with varying sparsity and size and using different number of threads (for CPUs) and block dimensions (for Nvidia GPUs). These data will then by used in the fusion algorithm for adaptive fusibility checks. For example, on one of our CPU platforms, `CostModel` finds 6-qubit dense gates take significantly longer time to simulate than 5-qubit dense gates. So it decides to limit the operation count per gate to 4096 (equaling to that of a 5-qubit dense gate) while allowing fused gates to act on up to 7 qubits. As shown in Table III, in a sample Random Quantum Circuit (RQC), the adaptive configuration achieves a comparable compression ratio with a significantly reduced operation count.

TABLE III
FUSION CONFIGURATION COMPARISON.

Configuration	Compression Ratio	Fusion Time (ms)	Operation Count
Adaptive	22.1x	1423	1.05M
Size-only	23.2x	988	1.74M
Non-agglomerative	23.2x	6660	3.28M

V. DYNAMIC KERNEL GENERATION AND COMPILATION

A. Automated Kernel Generation

`CAST` automates the kernel generation process for gates with various sizes and target qubits. Each `CAST` CPU kernel takes 4 arguments and each GPU kernel takes 2 arguments. These arguments are summarized in the following list:

1) Pointer to the statevector array.
2) Pointer to the matrix array. This is to support simulating parametrized quantum circuits.
3) Loop counter begin index, present only in CPU kernels.
4) Loop counter end index, present only in CPU kernels.

`CAST` backend kernel generator starts by emitting LLVM IR. In CPU kernels, the loop counter runs in $[0, 2^{n-k-s})$, where n is the number of qubits, k is the gate size, and $S = 2^s$ specifies vector size to be used in CPU-based SIMD instructions. This range can then be distributed to different threads for parallel execution. GPU kernels do not take loop counter as arguments because the loop counter is effectively tracked via built-in `blockIdx` and `threadIdx` variables.

Each GPU thread has an effective vector size of 1. So the following IR generation process also works for GPUs with $s = 0$.

For a k-qubit gate, we need to update 2^{k+1} vector amplitude registers in the main loop. The address and contiguity pattern of the amplitudes heavily depend on SIMD size and the set of target qubits.

Loading amplitude registers has two sub-steps: first calculate the start index from loop counter t, then depending on amplitude format, load 2^k real and imaginary amplitude registers each. We first split target qubits into two sets, the higher set and the lower set. Start by enumerating indices from 0. Color target qubit indices blue and color the remaining s smallest numbers red. Everything smaller than or equal to the largest red index is on the lower side (as shown in Fig. 7).

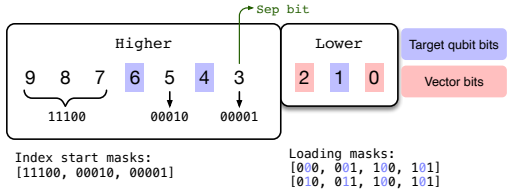


Fig. 7. Illustration when $s = 2$ and target qubits are $\{1, 4, 6\}$.

Let Q be the set of target qubits, H be the set of all higher indices, and L be the set of all lower indices. For some convenient numbers, let $k = |Q|$ be the size of the gate, and let $k_L = |Q \cap L|$ and $k_H = |Q \cap H|$ be the number of target qubits inside lower and higher set respectively. Then

- If $k_L = 0$, no runtime shuffling is needed both in loading and storing.
- Otherwise, the loop body loads (and stores) a total of 2^{k_H+1} length-2^{k_L+s} LLVM vectors. These vectors will then be shuffled to 2^{k+1} length-2^s vectors before matrix-vector multiplication. So on the LLVM IR level, when k_L is non-zero, shuffled loading works by loading larger LLVM registers and let the assembler split these registers and assign hardware-native shuffling instructions.
- Depending on hardware support, at most $3k_H + 1$ operations are needed to compute the start index:

$$\texttt{startIdx} = \sum_{i=0}^{|H \cap Q|} \texttt{(t \& masks[i]) << i}, \tag{4}$$

where `masks` is a compile-time known array[1].

B. JIT Execution and Re-compilation

As part of design flexibilities, `CAST` provides two modes for simulation on CPU-based platforms: JIT mode and static mode. JIT mode adopts LLVM ORC JIT engine to JIT-compile generated IR to native machine code for immediate execution. The static mode separate kernel generation and execution. It caches and saves generated IR files along with a header file of metadata. The saved IR files and metadata can then experience a compilation and linking phase to generate the final executable when needed.

GPU-based platforms follow a similar approach. The difference is that CUDA execution is internally JIT. `CAST` provides the flexibility of choosing to load the generated PTX files into a CUDA context in runtime or saving it to the disk for future needs.

[1]For machines that support BMI2 instruction set, a single runtime PDEP operation can replace Eq. (4). This is automatically configured in `CAST` argument `-use-pdep`.

VI. Evaluation

A. Setup

Software In our evaluations we used the following software:

- LLVM project: version 19.1.0.
- Qiskit: version 1.2.4 with backend QiskitAer 0.15.1.
- Qibo: version 0.2.13; QiboJIT: version 0.1.7.
- Cirq: version 1.4.1; QSimCirq: version 0.21.0.
- Qulacs: version 0.6.11.

Benchmark Circuits We used six types of benchmark circuits in our evaluations.

- Quantum Fourier Transform (QFT) [18].
- Alternating Layered Ansatz (ALA) [19].
- Random Quantum Circuit (RQC) [3].
- Quantum Volume Circuit (QVC) [20].
- Instantaneous Quantum Polynomial (IQP) [6].
- Hamiltonian Evolution Simulation (HES) [21].

Among the six types of circuits, QFT, RQC, IQP, and HES are relatively more sparse. So we call them *sparse-class* benchmarks. Correspondingly, ALA and QVC consist of mostly dense single-qubit gates, so we call them *dense-class* benchmarks.

B. Ablation Study on Adaptive Gate Fusion

To evaluate the performance improvement brought by our adaptive fusion algorithm, we perform an ablation study on different benchmark circuits with three fusion configurations: no fusion, preset size-only fusion, and adaptive fusion. We record the running time of each experiment under each fusion configuration and normalize it by the running time taken under the no fusion configuration.

As shown in Fig. 8, compared with the original circuit, size-only fusion optimization can reduce the running time by 68.7% $\sim$ 93.3% (resp. 62.0% $\sim$ 91.9%) over different benchmarks on the CPU (resp. GPU) platform. Compared with traditional size-only fusions, our new adaptive fusion algorithm obtains comparable performance on dense-class benchmarks, and can further reduce the running time by 28.7% $\sim$ 40.4% (resp. 18.5% $\sim$ 36.6%) on sparse-class benchmarks on the CPU (resp. GPU) platform.

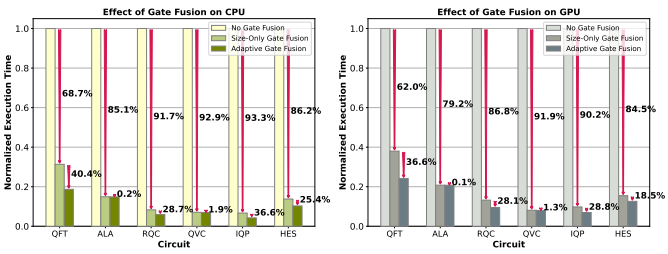


Fig. 8. Execution time of quantum simulation with different fusion strategies.

C. Cross-platform Backend

To demonstrate the effectiveness of `CAST` kernel generators on various platforms and various precisions, we benchmark and compare the performance of dynamic kernels generated by `CAST` and traditional ahead-of-time kernels as in Qiskit, Qibo, QSim, and Qulacs on the task of simulating gates with various sparsity on 28-qubit statevectors.

As reported in Table IV, our `CAST` backend achieves comparable performance on dense two-qubit and three-qubit gates compared with QSimCirq on AVX-512 platform `f32` mode and outperforms Qiskit, Qibo, QSim, and Qulacs on other benchmarks. This demonstrates our dynamically generated `CAST` backend has great versatility over different CPU and GPU platforms and simulation precisions. In addition, our backend generally runs faster on sparser gates, which

TABLE IV
COMPARISON OF SINGLE GATE PERFORMANCE.

Gate	Apple M2 (NEON)						Intel(R) Core(TM) i9-11900K (AVX512)						Nvidia RTX3090 (CUDA)					
	f32 Mode (GiBps)			f64 Mode (GiBps)			f32 Mode (GiBps)			f64 Mode (GiBps)			f32 Mode (GiBps)			f64 Mode (GiBps)		
	Qiskit	Qibo	Ours	Qiskit	Qibo	Ours	Qiskit	QSim	Ours	Qiskit	QSim	Ours	Qiskit	Qulacs	Ours	Qiskit	Qulacs	Ours
$H^{\otimes 2}$	1.74	3.20	**32.9**	3.46	6.04	**34.1**	1.28	15.6	**17.3**	2.50	N/A	**17.1**	75.2	N/A	**416**	181	150	**417**
$U3^{\otimes 2}$	1.74	3.18	**20.7**	3.45	5.87	**21.1**	1.28	15.5	**16.5**	2.62	N/A	**16.2**	75.4	N/A	**418**	180	151	**320**
$H^{\otimes 3}$	0.666	3.21	**22.0**	1.22	5.95	**22.8**	0.830	12.5	**15.6**	1.56	N/A	**14.9**	18.9	N/A	**396**	80.6	81.4	**308**
$U3^{\otimes 3}$	0.663	3.11	**8.38**	1.28	6.11	**8.50**	0.838	**12.4**	12.3	1.56	N/A	**12.7**	18.9	N/A	**393**	80.6	81.3	**149**

verifies CAST is able to automatically explore the sparsity pattern of gates.

D. Compilation Overhead

Compared with traditional approaches, CAST adopts extra code generation and re-compilation phases. We evaluate the impact of software overhead in this set of experiments.

We perform end-to-end simulations of RQCs with various number of qubits and record the normalized time spent in CAST front-end and hardware backend respectively. The front-end consists of the full process from parsing input files to obtaining executable. Normalized time is calculated by dividing the total end-to-end time spent with default fusion preset by the number of gates in the original circuit. As reported in Fig. 9, the ratio of CAST front-end overhead diminishes quickly as number of qubits increases. For example, in the 32-qubit experiment, the front-end of CAST only takes 0.59% (resp. 3.11%) of the end-to-end running time under the default (resp. adaptive) fusion configuration. This demonstrates CAST has great scalability in simulations. On the other hand, this set of experiments also suggests our adaptive fusion scheme exchanges slight front-end overhead for a great performance gain in running time.

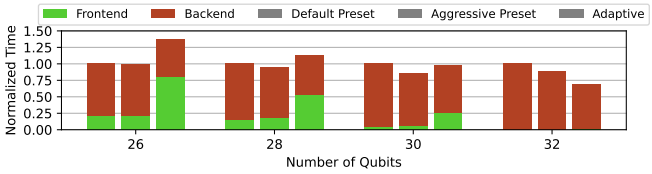


Fig. 9. Front-end overhead.

E. Full Circuit Simulation

We benchmark our CAST on full circuit simulations on CPU and GPU platforms.

CPU Performance We benchmark our simulator against Qiskit, Qibo, and QSimCirq on 32-qubit circuit simulations with multi-threading enabled. We record the execution time as reported by each simulator. Results are presented in Table V.

For single-precision mode simulations, compared with QSimCirq, our simulator achieves comparable performance on dense-class circuits and up to 3.68x speedup on sparse-class circuits such as QFT-30. This indicates our new sparsity-aware gate fusion algorithm is particularly effective in sparse circuits while still attaining a top performance on dense circuits. Over different types of benchmarks, CAST achieves 3.58x $\sim$ 8.03x (resp. 2.14x $\sim$ 4.92x) speedup than Qiskit and 3.60x $\sim$ 17.4x (resp. 2.29x $\sim$ 12.0x) speedup than Qibo in single-precision (resp. double-precision) mode simulations.

GPU Performance We benchmark our simulator against Qiskit and Qulacs on 30-qubit circuit simulations. We record the execution time reported in each software. Qulacs has its own CUDA backend that only supports double-precision simulations. The GPU backend of Qiskit is cuQuantum [22], which is developed by Nvidia and supports both f32 and f64 format simulation (Table I).

TABLE V
COMPARISON OF CPU PERFORMANCE.

Circuit	AMD EPYC 7543 (32-core, AVX2)							
	Single Precision (s)				Double Precision (s)			
	Qiskit	Qibo	QSim	Ours	Qiskit	Qibo	QSim	Ours
QFT-32	145	427	98.8	**27.4**	190	574	N/A	**51.4**
ALA-32	876	1011	204	**176**	956	1342	N/A	**337**
RQC-32	190	237	71.1	**26.8**	205	304	N/A	**55.8**
QVC-32	409	430	**110**	113	467	572	N/A	**223**
IQP-32	589	327	82.1	**25.3**	1087	420	N/A	**49.6**
HES-32	1918	1867	605	**201**	3457	2176	N/A	**408**

TABLE VI
COMPARISON OF GPU PERFORMANCE.

Circuit	Nvidia RTX3090					
	Single Precision (s)			Double Precision (s)		
	Qiskit	Qulacs	Ours	Qiskit	Qulacs	Ours
QFT-30	16.4	N/A	**2.44**	12.0	52.4	**5.44**
ALA-30	131	N/A	**9.83**	99.2	96.6	**28.3**
RQC-30	307	N/A	**15.9**	220	283	**79.7**
QVC-30	198	N/A	**5.04**	161	53.1	**25.5**
IQP-30	41.0	N/A	**2.01**	45.2	52.4	**5.08**
HES-30	683	N/A	**81.4**	479	1260	**173**

As shown in Table VI, over different types of benchmarks, CAST achieves 6.71x $\sim$ 39.3x speedup compared with Qiskit in single-precision mode simulations, and 2.08x $\sim$ 10.3x speedup compared with Qiskit and Qulacs in double-precision mode simulations.

The results demonstrate our toolchain has great performance and versatility in both single and double-precision modes and on different hardware platforms.

VII. CONCLUSION

This paper proposes a high-performance cross-platform quantum circuit simulation framework CAST (Cross-platform Adaptive Schrödinger-style Simulation Toolchain). CAST automates circuit optimization and kernel generation process by introducing a novel adaptive gate fusion and optimization algorithm and cross-platform backend kernel generators. Extensive evaluations suggest CAST can achieve higher performance over the leading industrial and academic quantum circuit simulators on large-size quantum circuit simulations with a minimized compilation overhead.

VIII. ACKNOWLEDGMENT

The support of UK EPSRC (grant number EP/W032635/1, EP/V028251/1, EP/S030069/1 and EP/X036006/1), Intel and AMD is gratefully acknowledged.

REFERENCES

[1] J. Tian, X. Sun, Y. Du, S. Zhao, Q. Liu, K. Zhang, W. Yi, W. Huang, C. Wang, X. Wu *et al.*, "Recent advances for quantum neural networks in generative learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 10, pp. 12 321–12 340, 2023.

[2] K. Blekos, D. Brand, A. Ceschini, C.-H. Chou, R.-H. Li, K. Pandya, and A. Summer, "A review on quantum approximate optimization algorithm and its variants," *Physics Reports*, vol. 1068, pp. 1–66, 2024.

[3] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, pp. 595–600, 2018.

[4] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[5] Y. Liu, X. Liu, F. Li, H. Fu, Y. Yang, J. Song, P. Zhao, Z. Wang, D. Peng, H. Chen *et al.*, "Closing the "quantum supremacy" gap: achieving real-time simulation of a random quantum circuit using a new sunway supercomputer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–12.

[6] M. J. Bremner, A. Montanaro, and D. J. Shepherd, "Achieving quantum supremacy with sparse and noisy commuting quantum computations," *Quantum*, vol. 1, p. 8, 2017.

[7] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio *et al.*, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.

[8] N. Hatano and M. Suzuki, "Finding exponential product formulas of higher orders," in *Quantum annealing and other optimization methods*. Springer, 2005, pp. 37–68.

[9] S. Aaronson and L. Chen, "Complexity-theoretic foundations of quantum supremacy experiments," *arXiv preprint arXiv:1612.05903*, 2016.

[10] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024.

[11] Q. A. team and collaborators, "qsim," Sep. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4023103

[12] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[13] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "Quest and high performance simulation of quantum computers," *Scientific reports*, vol. 9, no. 1, p. 10736, 2019.

[14] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. Garcia-Saez, J. I. Latorre, and S. Carrazza, "Qibo: a framework for quantum simulation with hardware acceleration," *Quantum Science and Technology*, vol. 7, no. 1, p. 015018, 2021.

[15] Y. Suzuki, Y. Kawase, Y. Masumura, Y. Hiraga, M. Nakadai, J. Chen, K. M. Nakanishi, K. Mitarai, R. Imai, S. Tamiya *et al.*, "Qulacs: a fast and versatile quantum circuit simulator for research purpose," *Quantum*, vol. 5, p. 559, 2021.

[16] M. Dou, T. Zou, Y. Fang, J. Wang, D. Zhao, L. Yu, B. Chen, W. Guo, Y. Li, Z. Chen *et al.*, "Qpanda: high-performance quantum computing framework for multiple application scenarios," *arXiv preprint arXiv:2212.14201*, 2022.

[17] J.-S. Kim, A. McCaskey, B. Heim, M. Modani, S. Stanwyck, and T. Costa, "Cuda quantum: The platform for integrated quantum-classical computing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–4.

[18] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE, 1994, pp. 124–134.

[19] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, and P. J. Coles, "Cost function dependent barren plateaus in shallow parametrized quantum circuits," *Nature communications*, vol. 12, no. 1, p. 1791, 2021.

[20] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, no. 3, p. 032328, 2019.

[21] G. H. Low and I. L. Chuang, "Optimal hamiltonian simulation by quantum signal processing," *Physical review letters*, vol. 118, no. 1, p. 010501, 2017.

[22] J. Faj, I. Peng, J. Wahlgren, and S. Markidis, "Quantum computer simulations at warp speed: Assessing the impact of gpu acceleration," *arXiv preprint arXiv:2307.14860*, 2023.

SnapPix: Efficient-Coding–Inspired In-Sensor Compression for Edge Vision

Weikai Lin[1,*], Tianrui Ma[2,3,*], Adith Boloor[3], Yu Feng[4,#], Ruofan Xing[5], Xuan Zhang[3,5], Yuhao Zhu[1]
[1]University of Rochester [2]Institute of Computing Technology, CAS [3]Washington University in St. Louis
[4]Shanghai Jiao Tong University [5]Northeastern University
wlin33@ur.rochester.edu, matianrui@ict.ac.cn, adith@wustl.edu, y-feng@sjtu.edu.cn,
{xing.ruo, xuan.zhang}@northeastern.edu, yzhu@rochester.edu

***Abstract*—Energy-efficient image acquisition on the edge is crucial for enabling remote sensing applications where the sensor node has weak compute capabilities and must transmit data to a remote server/cloud for processing. To reduce the edge energy consumption, this paper proposes a sensor-algorithm co-designed system called SnapPix, which compresses raw pixels in the analog domain inside the sensor. We use coded exposure (CE) as the in-sensor compression strategy as it offers the flexibility to sample, i.e., selectively expose pixels, both spatially and temporally. SnapPix has three contributions. First, we propose a task-agnostic strategy to learn the sampling/exposure pattern based on the classic theory of efficient coding. Second, we co-design the downstream vision model with the exposure pattern to address the pixel-level non-uniformity unique to CE-compressed images. Finally, we propose lightweight augmentations to the image sensor hardware to support our in-sensor CE compression. Evaluating on action recognition and video reconstruction, SnapPix outperforms state-of-the-art video-based methods at the same speed while reducing the energy by up to 15.4×. We have open-sourced the code at: https://github.com/horizon-research/SnapPix.**

I. Introduction

Energy-efficient imaging is essential for sensing scenarios where the sensor node has weak compute capability and must transmit data to a nearby edge server or even cloud for processing. This includes scenarios such as traffic monitoring [1], remote satellite sensing [2], or even smartphone/extended reality devices that offload computations [3]. In all these scenarios, the sensing node energy is dominated by two components: 1) the image sensor itself, which is in turn dominated by the readout circuitry [4], [5] , and 2) the wireless data transmission from the sensing node, whose energy is known to outweigh computation and increases with transmission distance [6].

To address this energy bottleneck, recent research has explored in-sensor compression, where raw pixels are aggressively sampled before being digitized and transmitted out of the sensor [4], [5], [7]–[9]. While promising, in-sensor compression faces two challenges.

- First, it must address a fundamental dilemma, where the sensor output is both an *energy* bottleneck and an *information* bottleneck, which are directly in contention with each other. Reducing energy consumption requires aggressive in-sensor compression, but over-compression can lead to significant losses in task accuracy.
- Second, the compressed data must support diverse downstream tasks. For instance, a smart-city camera might need to process the same video stream for both action recognition and traffic monitoring. Similarly, in augmented reality (AR) applications, a single camera stream must handle tasks such as hand recognition and scene reconstruction [10].

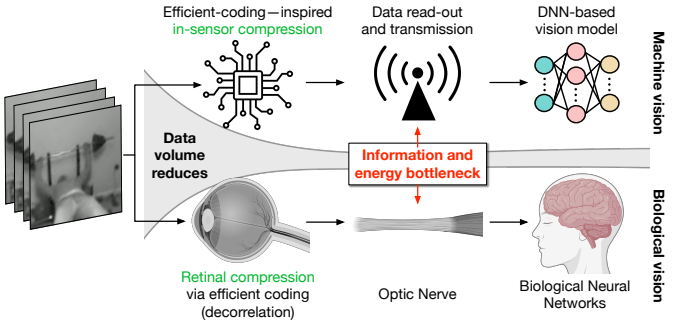


Fig. 1. SnapPix reduces edge sensing energy through in-sensor compression by decorrelating output pixel values. This is inspired by the mammalian visual system, where the retina compresses information by decorrelating the retinal output neurons; signals carried through the optic nerve, while at a much lower bandwidth than at the initial stage of the retina, encodes essential information that permits the downstream visual cortex to effectively perform visual tasks.

The paper proposes SnapPix, a general in-sensor compression architecture that significantly reduces end-to-end system energy while maintaining high accuracy. SnapPix achieves this through three key innovations, summarized below.

General-Purpose Sampling via Decorrelation. To design a general compression strategy, SnapPix leverages the principle of decorrelation to minimize redundancy among output pixels (Sec. III). This approach is inspired by the classic theory of efficient coding in neuroscience [11]–[14], which suggests that the retina efficiently transmits information to the brain by reducing redundancy and increasing decorrelation among the Retina Ganglion Cells (RGCs) — the output neurons of the retina [15]. This concept is visualized in Fig. 1.

Similarly, the compression pattern in SnapPix is trained to maximize information density in the sensor output, as opposed to be tailored to a particular task. The core mechanism for

*Both authors contributed equally to this research.
#Work done while at University of Rochester.

compression in SNAPPIX is *sampling*. In particular, SNAPPIX samples through *coded exposure* (CE) [16], where the sensor selectively exposes pixels both spatially and temporally; pixel values are integrated across exposure slots into a single coded image before being read out [7].

Co-Designed Vision Model and Coded Pattern. To maximize the benefits of CE, ideally there should not be any constraint imposed on each pixel's exposure pattern. That, however, means that pixels would carry varying amounts of information and should be treated differently in the downstream model, introducing performance overhead [17], [18].

Instead, SNAPPIX constrains the sampling pattern to be tile-repetitive — pixels within a tile can have different exposure patterns, but the pattern repeats across tiles — and thus constrains the pixel variation within a tile (Sec. IV). To accommodate this tile-repetitive structure, we use Vision Transformers (ViTs) [19] as the backbone. ViTs naturally process inputs tile-by-tile, and the processing of each tile is trained based on the (offline obtained) within-tile pixel variations. With a carefully designed ViT architecture and tailored pre-training, our constrained sampling does not degrade accuracy.

Hardware Support. To support our in-sensor sampling strategy, we propose a set of minimal hardware augmentations to the stacked sensor architecture that is common in modern image sensors [20], where the first layer is the pixel array and the second layer implements per-pixel digital logic (Sec. V). The augmentations to each pixel are limited to a single-bit storage and two transistors to save and apply the CE pattern. These augmentations are integrated beneath the pixel array, resulting in negligible area overhead.

Results. On various edge sensing scenarios (that differ in edge compute capabilities and in data transmission technologies), SNAPPIX achieves 1.4× to 15.4× energy saving compared to existing systems. Compared to other compression methods with a similar compression rate, SNAPPIX delivers better accuracy on multiple tasks.

II. BACKGROUND

A. Energy Bottleneck in Edge Sensing

This paper examines edge sensing scenarios where the edge comprises an image sensor and a lightweight compute unit, such as a microcontroller. In these setups, data must be offloaded to a server for computation-intensive processing. This paradigm is common in remote sensing applications, including urban sensing [21] and satellite imaging [2].

The main energy bottleneck in such systems comes from the cost of in-sensor data read-out and sensor-host data transmission. The read-out energy cost is dominated by the ADC. A recent survey shows that the ADC costs about 66% of an image sensor's energy on average [5]. The data transmission includes both the MIPI CSI-2 interface, which transfers data from the sensor to the edge processor, and the subsequent transmission of data from the processor to the cloud. Energy-wise, transmitting one byte via the MIPI CSI-2 interface is approximately 300 times more expensive than performing a one-byte MAC operation [22]. Wireless transmission exacerbates energy consumption, adding an order of magnitude to the cost, even with passive transmission methods like backscatter [23].

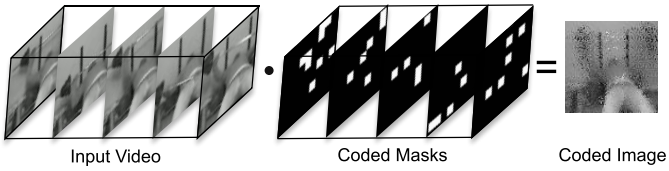


Fig. 2. Coded exposure with 5 exposure slots. In each slot, pixels are selectively exposed, controlled by a coded mask. In the end, the values at all the exposure slots are integrated pixel-wise to form one single coded image.

B. Coded Exposure (CE)

CE compresses image data by selectively exposing pixels across spatial dimensions and multiple frames, integrating the exposures into a single coded image before readout [7]. Fig. 2 illustrates the fundamental operation of CE. Let Y represent the sequence of images that would have been captured using a conventional image sensor, with dimensions $T \times H \times W$, where T is the number of frames in Y, and H and W are the height and width of each image, respectively.

In the CE process, an exposure mask is applied to each frame (or exposure slot) t to select a subset of pixels to expose during that frame (e.g., 5 exposure slots as shown in Fig. 2). The exposed pixel values are then integrated over time for each pixel, resulting in a single coded image of dimensions $1 \times H \times W$ being read out from the sensor. This approach achieves a data reduction factor of T, as T frames are compressed into one. This process can be formulated as follows:

$$X(i,j) = \sum_{t=1}^{T} M(i,j,t) \cdot Y(i,j,t) \tag{1}$$

where X is the final coded image, M is the binary masks controlling exposure, i, j, and t index the spatial dimensions and the temporal dimension (frames), respectively.

III. GENERAL-PURPOSE SAMPLING VIA DECORRELATION

Our CE pattern design maximizes decorrelation among pixels in coded images. Proximal pixels often exhibit high similarity and strong correlation, while pixels farther apart tend to be less correlated. To focus on reducing redundancy among highly correlated pixels, we decorrelate pixels within a tile of P pixels. This is achieved by minimizing the following loss:

$$\mathcal{L}_{\text{Cor}} \triangleq \frac{1}{P(P-1)} \sum_{i=1}^{P} \sum_{j \neq i} C_{ij}^2 \tag{2}$$

where C_{ij} is the Pearson correlation coefficient between two distinct coded pixels i and j within a tile, quantifying redundancy between them.

Estimating C_{ij} requires multiple samples for each coded pixel, and Fig. 3 illustrates this process. First, we sample a batch of images from the dataset and apply the CE operation (Eqn. 1) to generate a batch of coded images. Then we divide each coded image into $N \times N$ tiles, yielding $S = B \times N^2$ samples per coded pixel, where B is the number of coded

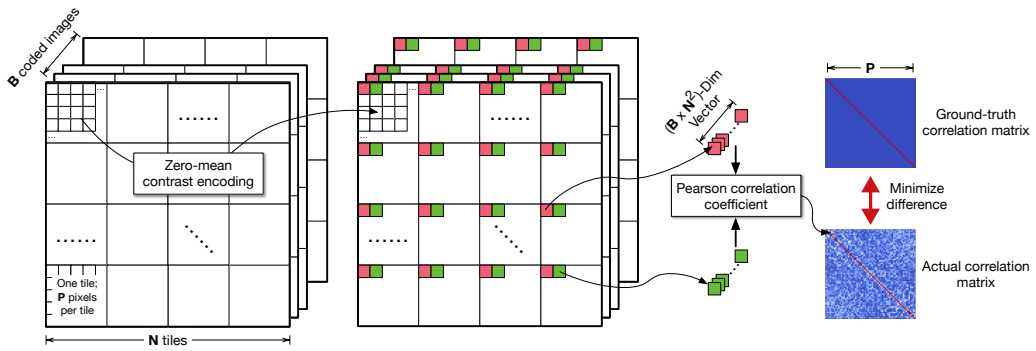


Fig. 3. Illustration of training for pixel decorrelation in coded images. A coded image is divided into tiles, each containing P pixels. The CE mask (not shown) is optimized to decorrelate any pair of pixels within a tile. Zero-mean contrast encoding is applied, ensuring the mean pixel value of each tile is zero.

images in one batch and N^2 is the number of tiles per coded image. As a result, each coded pixel in the batch is represented by an S-dimensional vector, which is used to calculate C_{ij} for any pair of coded pixels i and j.

Before forming the S-dimensional vectors, we preprocess each raw coded tile to have a zero mean, a step referred to as "zero-mean contrast encoding" in Fig. 3. Mathematically, for each tile in a raw coded image, we subtract the average pixel value of the tile from every pixel in the tile. The average value is computed by averaging across all the corresponding tiles in a dataset. This zero-mean preprocessing is important, as proximal pixels naturally exhibit similar values and are thus highly correlated. Without accounting for this inherent correlation, the training process can collapse, where all exposure slots remain closed for all pixels.

We learn the optimal exposure mask (M in Eqn. 1) by minimizing $\mathcal{L}_{Cor}$. This training is task-agnostic: the exposure mask is optimized for a given dataset irrespective of the downstream task. We use straight-through estimation [24] to propagate gradients through the binary masking operation.

IV. CE-Optimized Vision Model and Training

Motivation. Pixels in coded images carry varying amounts of information due to their differences in exposure. A pixel exposed in more slots retains richer temporal information and carries more motion blur. Downstream models that ignore these variations experience accuracy degradation.

In particular, standard convolution operations treat all pixels uniformly, applying the same kernel to every pixel regardless of its exposure. Prior work [17] proposed Shift-Variant Convolution (SVC) layer to address this issue by using different convolution kernels for different coded pixels. However, our profiling shows that SVC slows inference by 4× due to a lack of optimized implementations and inefficient GPU utilization. Due to this performance issue, SVC has only been applied to the first layer in previous work [17], [18], negatively impacting the overall accuracy.

CE-Optimized Vision Transformer. If the exposure pattern is allowed to vary across all the pixels in a frame, then all the pixels are necessarily different. Instead, we propose a tile-repetitive exposure pattern: the frame is divided into tiles, with pixel exposure variations allowed within each tile but constrained to be identical across all tiles. As shown in the left part of Fig. 4: the exposure pattern is consistent across tiles for each time slot (three slots are illustrated), but it can change across slots. As a result, the downstream models only have to deal with pixel variations within a tile while maintaining performance comparable to unconstrained patterns [18].

Building on this tile-repetitive pattern, we propose using Vision Transformers (ViTs) [19] as the backbone for the downstream vision model. ViTs naturally divide an image into patches of $M \times M$ pixels and process pixels within each patch differently via patch-wise embedding (PE) and multi-layer perceptrons (MLPs), while enabling information sharing across patches through multi-head attention (MHA). This makes ViTs well-suited for handling localized pixel variations, as illustrated in the middle part of Fig. 4.

We set the CE tile size to match the ViTs' patch size. This ensures MLPs can learn to address pixel-wise variations within each tile, which are determined by the decorrelation-based exposure pattern. To further facilitate ViT training, each pixel value is normalized by the number of exposure slots.

CE-Optimized Reconstruction Pre-training. Inspired by recent reconstruction-based pre-training [25], [26], we propose a tailored pre-training procedure for CE-compressed inputs.

As illustrated in the right part of Fig. 4, the pre-training process begins with a CE-coded image X, sampled and integrated from a sequence of images. We randomly mask a large portion (e.g., 85%) of its tiles, and set the objective of pre-training to reconstruct the original video sequence. Unlike previous image-to-image [25] or video-to-video [26] pre-training approaches, our pre-training is designed to perform a "coded image-to-video" prediction. This requires the model to both predict the masked tiles, capturing spatial scene structure, and upsample temporal signals from CE-coded information, learning temporal dynamics.

The pre-training process can be formulated as follows:

$$\hat{Y} = \mathrm{D}(\mathrm{E}(\text{random_masking}(f(Y)))) \tag{3}$$

where Y is the original video, $f(\cdot)$ is CE function as defined in Eqn. 1, $E(\cdot)$ and $D(\cdot)$ are the pre-trained ViT encoder and decoder, and $\hat{Y}$ is the reconstructed video. We use the Mean Squared Error (MSE) loss for pre-training, and predict only 50% of the video frames to accelerate pre-training [26].

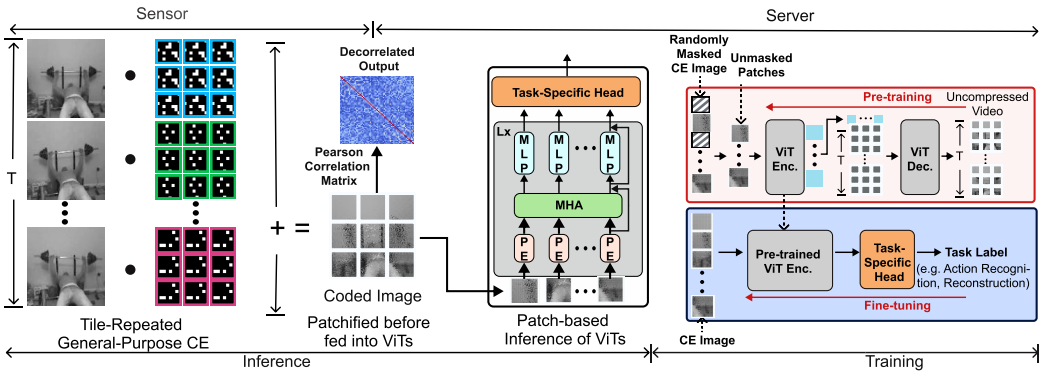


Fig. 4. The end-to-end pipeline of SnapPix with in-sensor CE for compression and a ViT-based vision model for downstream tasks. The CE pattern is trained task-independently using decorrelation, while the downstream model is co-designed with CE patterns and pre-trained specifically for CE-encoded inputs.

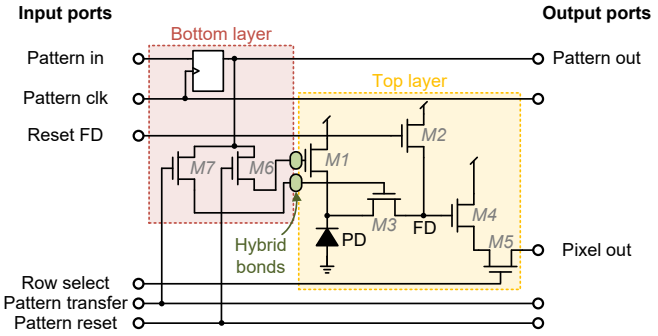


Fig. 5. Schematic of the proposed CE pixel. It is based on a stacked design that is commonly used in modern CMOS image sensors [20].

V. Hardware Support for In-Sensor CE

Rationale and Overview. To support the CE function, previous solutions heavily augment each pixel, placing multiple floating diffusion (FD) nodes [27], [28], using an arithmetic logic unit with mixed-signal memory [29], or communicating high-resolution CE pattern with off-chip controller [30]. They result in a larger pixel area and a smaller pixel fill factor.

In contrast, our sensor utilizes the characteristic of the tile-repetitive decorrelation pattern to reduce both the control footprint and control power. In our design, we replace the global control with local, per-pixel storage to store the CE pattern. We then use a die stacking design, where the top layer is the pixel array and the bottom layer hosts the digital logic/storage. Such a stacking design is common in consumer CMOS image sensors [20], and our design represents a novel usage of the bottom layer to support CE. By stacking, the per-pixel storage is completely hidden beneath the pixel's exposure circuits, so the area of our proposed pixel is approximately the same as that of the conventional, non-CE pixel.

Design. The proposed pixel design is illustrated in Fig. 5, which contains two layers. The top layer is pixel array layer, where each pixel is based on the classic 4T Active Pixel Sensor (APS) design that consists of a photodiode (PD) to acquire incident light and five transistors to reset PD (`M1`), reset floating diffusion (FD) node (`M2`), transfer photocharge (`M3`), and read out photocharge as voltage when the pixel is selected (`M4`, `M5`). Different to the conventional 4T pixel design, the additional transistor `M1` decouples the reset of PD and FD such that the PD can be exposed across multiple exposure slots but only transfer the photocharge once to the FD, thereby realizing CE function.

At the bottom layer, each pixel is equipped with a D-Flip-Flop (DFF) to buffer the one-bit CE pattern of a given exposure slot. The bit represents whether to accumulate the pixel's exposure during the corresponding exposure slot. The DFFs of all the pixels in a tile are connected in a shift-register style: the *pattern in* wire of the second pixel is connected to the *pattern out* wire of its preceding pixel. Each pixel is also augmented with two additional transistors: `M6` (pattern reset) and `M7` (pattern transfer).

At the start of every exposure slot, the CE bits of each pixel in a tile are streamed in through the *pattern in* wire and buffered in the corresponding DFFs . Then, `M6` is turned on for all the pixels (through the *pattern reset* wire), which allows the CE bit in the DFF to control the reset of the PD. If the CE bit is 1, the PD is reset via `M1` (charges accumulated so far is cleared), getting ready for exposure; if the CE bit is 0, the PD is not reset (since `M1` will be open). The DFFs can then be powered-gated.

After the exposure, the same CE bits are streamed in and buffered again in the DFFs. We then turn on `M7` for all the pixels (through the *pattern transfer* wire). If the CE bit is 1, the charge is transferred from the PD to the FD through `M3`; otherwise, `M3` is open and the FD does not accumulate the charge from the previous exposure. The DFFs are then power-gated again until the next exposure slot. While the DFFs are power-gated, logic 0 is given to both `M1` and `M3` via a simple reset logic (not shown in Fig. 5 for simplicity). The delay between the *pattern reset* and *pattern transfer* signals physically creates the exposure time for the pixel [7].

Area Overhead. We synthesize the digital logic at the bottom layer in TSMC 65 nm and obtain the area estimation of 30 μm^2. According to DeepScale tool [31], it translates to 3.2 μm^2 in 22 nm, which is much smaller than commercial stacked digital pixel sensors (DPS) [32], [33]. Therefore, the pixel area is constrained by the APS at the top layer rather than the introduced logic.

An alternative design is to broadcast the CE pattern to all

the pixels in a tiles via dedicated wires. This approach would remove the need for per-pixel digital logic, but require $2N$ signal wires per pixel given a tile size of $N \times N$. In contrast, the number of wires in our design is constant: only four (*pattern in*, *pattern clk*, *pattern transfer*, and *pattern reset*), regardless of tile size. Our synthesis results show that when $N = 8$, the signal wires occupy $2.24\ \mu m \times 2.24\ \mu m$; as N increases to 14, the wire area grows to $3.92\ \mu m \times 3.92\ \mu m$, exceeding the area of the state-of-the-art APS.

VI. Evaluation

A. Experimental Setup

Dataset. We evaluate our approach on three datasets: Something-Something v2 (SSV2) [34], Kinetics-400 (K400) [35], and UCF-101 [36], using the first train-test split in UCF-101. For pre-training, we use SSV2 and the larger K710 dataset [26]. We downsample each video's shorter dimension to 112 pixels and convert the videos to grayscale in linear space. We simulate a CE dataset using $T = 16$ (recall T is the number of exposure slots in Eqn. 1). Each frame is center-cropped to produce a 112×112 resolution input, which is also the input resolution to SnapPix. The video baselines accepts 16 consecutive, uncoded frames as their inputs (i.e., $16 \times 112 \times 112$ in resolution).

Tasks. We evaluate two distinct downstream tasks: action recognition (AR), a high-level task producing a single classification output, and reconstruction (REC), a low-level task generating a video. REC evaluation addresses scenarios where videos are stored for future, undefined tasks. AR is tested on SSV2, K400, and UCF101 using clip-1 crop-1 accuracy, while REC is evaluated on SSV2, which reconstructs the original $16 \times 112 \times 112$ frames from a 112×112 coded image, with Peak Signal-to-Noise Ratio (PSNR) as the metric.

Variants. We provide two SnapPix variants: SnapPix-B and SnapPix-S, differing only in the backbone. SnapPix-B uses ViT-B (87M parameters), for higher accuracy but at a slower speed. SnapPix-S uses ViT-S (22M parameters) for faster performance, similar to prior CE-based models [17], [18]. Both variants use an 8×8 patch size.

Baselines. We compare our decorrelated CE pattern with the following task-agnostic CE patterns ($T = 16$ in all cases):

- Long Exposure: All pixels exposed in all slots.
- Short Exposure: All pixels exposed every 8th frame.
- Random: Each pixel exposed randomly with a 50% probability per exposure slot.
- Sparse Random: Each pixel exposed once randomly across T exposure slots.

For AR, we also compare against three prior systems:

- SVC2D [17]: A CE-based AR model with SVC and an end-to-end learned CE pattern.
- C3D [37]: A strong video-based AR model, treated as an upper bound in prior CE works [17], [18].
- VideoMAEv2-ST [26]: A state-of-the-art ViT-based video AR model, adjusted to match SnapPix-B's speed.

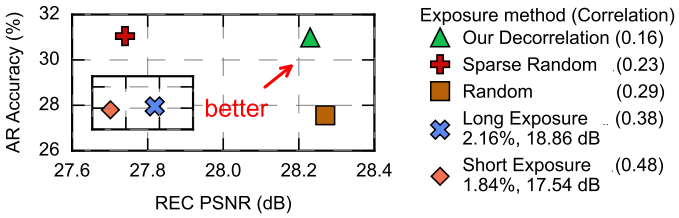


Fig. 6. Comparison of task-agnostic CE patterns using AR and REC results. The legend also shows the corresponding Pearson correlation coefficients.

To ensure a fair comparison, we reproduced all baselines using our data preprocessing, frame sampling strategy, and training recipe, modifying only the learning rate.

Training Recipe. To avoid overfitting to smaller datasets, we first train the decorrelated CE pattern on the large pre-training dataset for 5 epochs. We then fix the CE pattern and perform pre-training and task-specific training for downstream vision models. For training from scratch, REC is trained for 100 epochs, AR for 200 epochs on SSV2, and 400 epochs on K400 and UCF-101. For fine-tuning after pre-training, epochs are halved. Epochs are computed as repeated augmentations$\times$epochs, same as prior work [26]. Learning rates are tuned for different models.

B. Decorrelation Outperforms Other Task-Agnostic CEs

To evaluate the effectiveness of our decorrelation-based CE (Sec. III), we compare it against other task-agnostic CE patterns in Fig. 6, where reports the AR accuracy (y-axis) and the REC quality (x-axis) on the SSV2 dataset using our CE-optimized ViT (Sec. IV) trained from scratch. The legend on the right also reports the Pearson correlation coefficient between coded pixels.

Our decorrelation-based CE excels in both AR and REC, demonstrating its effectiveness as a general task-agnostic CE pattern. While Random performs slightly better in REC, its AR accuracy is significantly lower than ours. Conversely, Sparse Random achieves a high AR accuracy but performs poorly in REC. Long Exposure and Short Exposure perform significantly worse in both tasks. The relative performance across these strategies corresponds roughly to their respective correlation coefficient, suggesting the benefit of using decorrelation to learn the CE pattern.

C. SnapPix Outperforms Task-Specific Methods

We now evaluate SnapPix against prior systems that are specifically tuned for AR, including CE-based and video-based methods. Inference speed is measured in inference per second on a system with a Ryzen 9 7900X3D CPU and a RTX 4090 GPU using a batch size of 64, simulating server-side computations. Results are summarized in Tbl. I. SnapPix-B and SnapPix-S achieve the highest AR accuracy across all datasets, nearly doubling the performance of CE-based baselines. They also surpass video models, which were previously considered upper bounds for CE-based methods [17], [18]. At the same time, SnapPix executes faster than video-based methods due to a reduced input data.

TABLE I
COMPARISON WITH PREVIOUS SYSTEMS. WE HIGHLIGHT THE TOP 3 VARIANTS FOR EACH METRIC (RED, ORANGE, YELLOW).

Model	Input	Accuracy (↑)			Inference /sec (↑)
		UCF-101	SSV2	K400	
SNAPPIX-S (ours)	CE	74.65%	42.38%	47.58%	2282
SNAPPIX-B (ours)	CE	79.14%	45.21%	54.11%	760
SVC2D [17]	CE	41.16%	23.05%	26.09%	2135
C3D [37]	Video	62.70%	33.48%	41.66%	541
VideoMAEv2-ST [26]	Video	72.54%	39.84%	41.99%	750

It might initially seem surprising that SNAPPIX outperforms even the video baselines (last two rows in Tbl. I). This is primarily because SNAPPIX uses a single image as the model input, which significantly reduces the compute overhead and, in turn, allows us to use larger models.

SVC2D is a prior CE-based AR method that learns the CE pattern jointly with SVC in the downstream model [17], [18]. When we apply its CE pattern to our ViT model, it underperforms our decorrelation-based pattern by 1.35% in AR accuracy (not shown in the table). This highlights a key limitation of task-specific, end-to-end learned patterns: they are model-dependent and become suboptimal even for the same task when the model architecture evolves.

D. SNAPPIX *Significantly Reduces Edge Energy*

Our energy evaluation is mainly based on edge-server scenarios where the edge device has to transmit all the data to the cloud/server. In this case, the edge energy is the sum of the sensing and data transmission energy. We use CamJ [22], a sensor energy model calibrated against real silicon to model the sensing energy. The total sensing energy is 220 pJ per pixel (8 bits), of which 95.6% is contributed to by the ADC and MIPI energy [5]. The energy overhead introduced by supporting CE is 9 pJ per pixel with 20 MHz pattern stream clock according to our synthesis results.

The wireless energy depends on the technology used for data transmission. We model short-range (about 10 meters) transmission using passive WiFi, with a transmission energy of 43.04 pJ per pixel [38]. Long-range (over 100 meters) transmission usually uses backscatter Long-Range (LoRa) technology [23], whose energy is 7.4 μ J per pixel.

Under $T = 16$, SNAPPIX reduces the ADC/MIPI and wireless transmission energy by 16×. SNAPPIX achieves a 7.6× edge energy saving in the short-range scenario and a 15.4× saving in the long-range scenario.

We also evaluate a scenario where the edge node has a mobile GPU that can execute the downstream model. For that we measure the energy consumption of the mobile Volta GPU on a Jetson Xavier SoC [39] with a batch size of 1. In this case the mobile GPU energy dominates the total energy consumption. Compared to executing video baselines: VIDEOMAEV2-ST and C3D on the edge server, SNAPPIX-S achieves 1.4× and 4.5× energy saving respectively. This is primarily because SNAPPIX uses single (coded) images to drive the vision model rather than an entire video.

Finally, we compare with a simple compression baseline that spatially downsamples each frame by 16× (the same compression rate as SNAPPIX) using 4×4 average filtering and then processing the compressed data with VIDEOMAEV2-ST. The resulting AR accuracy is 9.83%, 6.24%, and 16.45% *lower* than SNAPPIX-B on UCF-101, SSV2, and K400, respectively.

E. Ablation Study

We perform an ablation study by removing various components from Sec. III and Sec. IV, using SNAPPIX-S as a baseline and the SSV2 dataset and AR task for evaluation:

- Removing pre-training reduces accuracy by 11.39%.
- Replacing the decorrelated pattern with a random pattern further decreases accuracy by 3.43%.
- Replacing the tile-repetitive CE pattern with a global pattern reduces accuracy by 23.74%.

VII. RELATED WORK

Compressed Edge Sensing. Digital-domain compression [40], [41] could achieve high compression rate. However, even with dedicated hardware, classic digital compression consumes nJ/pixel [42], several orders of magnitude higher than the energy of sensing itself. This issue is further exacerbated in deep-learning-based compression [41]. Moreover, digital compression operates after sensor read-out without sensing energy saving. Instead, in-sensor compression [4], [5], [7]–[9] performs compression before read-out, saving both sensing and transmission energy. Our method falls under the category of in-sensor compression.

Sensors with CE Support. Coded Exposure (CE) is a computational imaging technique used for enhancing imaging quality (e.g. High Dynamic Range (HDR) [27], [43] and high-speed imaging [7]). We propose a novel use of CE — for in-sensor compression. Algorithmically compared to prior CE-based methods [16]–[18], we are the first to propose using decorrelation to learn the CE patterns, and have shown that such a pattern out-perform prior methods across multiple tasks.

In-Sensor Computation. Vision sensors increasingly integrate computational capabilities, shifting computations to the analog domain. Examples include selective reading [4], [8], feature extraction [44], computing temporal derivatives [45], and even lightweight neural networks [5], [9], [46]. SNAPPIX supports in-sensor CE for compression, and introduces only minor augmentations to the hardware design.

VIII. CONCLUSION

In-sensor compression through CE, which turns a sequence of video frames into a single coded image, leads to an order of magnitude energy saving on edge sensing while maintaining a task quality comparable with existing video-based methods. Task-agnostic training of CE patterns through decorrelation can generalize to downstream tasks of different nature, outperforming other CE patterns.

ACKNOWLEDGMENT

The work is partially supported by NSF Award #2416375.

REFERENCES

[1] J. Klotz and S. K. Nayar, "Minimalist vision with freeform pixels," in *ECCV*, pp. 329–346, 2024.

[2] A. Gadre, Z. Machester, and S. Kumar, "Adapting lora ground stations for low-latency imaging and inference from lora-enabled cubesats," *TOSN*, vol. 20, no. 5, pp. 1–30, 2024.

[3] XREAL, "Xreal air 2." https://us.shop.xreal.com/products/xreal-air-2.

[4] Y. Feng, T. Ma, Y. Zhu, and X. Zhang, "Blisscam: Boosting eye tracking efficiency with learned in-sensor sparse sampling," in *ISCA*, pp. 1262–1277, 2024.

[5] T. Ma, A. J. Boloor, X. Yang, W. Cao, P. Williams, N. Sun, A. Chakrabarti, and X. Zhang, "Leca: In-sensor learned compressive acquisition for efficient machine vision on the edge," in *ISCA*, pp. 1–14, 2023.

[6] H. Desai, M. Nardello, D. Brunelli, and B. Lucia, "Camaroptera: A long-range image sensor with local inference for remote sensing applications," *TECS*, vol. 21, no. 3, pp. 1–25, 2022.

[7] M. Yoshida, T. Sonoda, H. Nagahara, K. Endo, Y. Sugiyama, and R.-i. Taniguchi, "High-speed imaging using cmos image sensor with quasi pixel-wise exposure," *TCI*, vol. 6, pp. 463–476, 2019.

[8] T. Zhang, K. Kasichainula, D.-W. Jee, I. Yeo, Y. Zhuo, B. Li, J.-s. Seo, and Y. Cao, "Improving the efficiency of cmos image sensors through in-sensor selective attention," in *ISCAS*, pp. 1–4, IEEE, 2023.

[9] G. R. Nair, P. S. Nalla, G. Krishnan, J. Oh, A. Hassan, I. Yeo, K. Kasichainula, M. Seok, J.-s. Seo, Y. Cao, *et al.*, "3d in-sensor computing for real-time dvs data compression: 65nm hardware-algorithm co-design," *SSC-L*, 2024.

[10] H. Kwon, K. Nair, J. Seo, J. Yik, D. Mohapatra, D. Zhan, J. Song, P. Capak, P. Zhang, P. Vajda, *et al.*, "Xrbench: An extended reality (xr) machine learning benchmark suite for the metaverse," *MLSys*, vol. 5, pp. 1–20, 2023.

[11] H. B. Barlow *et al.*, "Possible principles underlying the transformation of sensory messages," *Sensory communication*, vol. 1, no. 01, pp. 217–233, 1961.

[12] F. Attneave, "Some informational aspects of visual perception.," *Psychological review*, vol. 61, no. 3, p. 183, 1954.

[13] X. Pitkow and M. Meister, "Decorrelation and efficient coding by retinal ganglion cells," *Nature neuroscience*, vol. 15, no. 4, pp. 628–635, 2012.

[14] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *ICML*, pp. 12310–12320, PMLR, 2021.

[15] Y. Dan, J. J. Atick, and R. C. Reid, "Efficient coding of natural scenes in the lateral geniculate nucleus: experimental test of a computational theory," *Journal of neuroscience*, vol. 16, no. 10, pp. 3351–3362, 1996.

[16] D. Reddy, A. Veeraraghavan, and R. Chellappa, "P2c2: Programmable pixel compressive camera for high speed imaging," in *CVPR*, pp. 329–336, IEEE, 2011.

[17] T. Okawara, M. Yoshida, H. Nagahara, and Y. Yagi, "Action recognition from a single coded image," in *ICCP*, pp. 1–11, IEEE, 2020.

[18] S. Kumawat, T. Okawara, M. Yoshida, H. Nagahara, and Y. Yagi, "Action recognition from a single coded image," *TPAMI*, vol. 45, no. 4, pp. 4109–4121, 2022.

[19] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021.

[20] Y. Oike, "Evolution of image sensor architectures with stacked device technologies," *IT-ED*, vol. 69, no. 6, pp. 2757–2765, 2021.

[21] J. Adkins, B. Ghena, N. Jackson, P. Pannuto, S. Rohrer, B. Campbell, and P. Dutta, "The signpost platform for city-scale sensing," in *IPSN*, pp. 188–199, IEEE, 2018.

[22] T. Ma, Y. Feng, X. Zhang, and Y. Zhu, "Camj: Enabling system-level energy modeling and architectural exploration for in-sensor visual computing," in *ISCA*, pp. 1–14, 2023.

[23] V. Talla, M. Hessar, B. Kellogg, A. Najafi, J. R. Smith, and S. Gollakota, "Lora backscatter: Enabling the vision of ubiquitous connectivity," *IMWUT*, vol. 1, no. 3, pp. 1–24, 2017.

[24] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[25] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *CVPR*, pp. 16000–16009, 2022.

[26] L. Wang, B. Huang, Z. Zhao, Z. Tong, Y. He, Y. Wang, Y. Wang, and Y. Qiao, "Videomae v2: Scaling video masked autoencoders with dual masking," in *CVPR*, pp. 14549–14560, 2023.

[27] R. Gulve, N. Sarhangnejad, G. Dutta, M. Sakr, D. Nguyen, R. Rangel, W. Chen, Z. Xia, M. Wei, N. Gusev, *et al.*, "39 000-subexposures/s dual-adc cmos image sensor with dual-tap coded-exposure pixels for single-shot hdr and 3-d computational imaging," *JSSC*, vol. 58, no. 11, pp. 3150–3163, 2023.

[28] Y. Luo and S. Mirabbasi, "A 30-fps 192× 192 cmos image sensor with per-frame spatial-temporal coded exposure for compressive focal-stack depth sensing," *JSSC*, vol. 57, no. 6, pp. 1661–1672, 2022.

[29] J. N. Martel, L. K. Mueller, S. J. Carey, P. Dudek, and G. Wetzstein, "Neural sensors: Learning pixel exposures for hdr imaging and video compressive sensing with programmable sensors," *TPAMI*, vol. 42, no. 7, pp. 1642–1653, 2020.

[30] K. Kagawa, M. Horio, A. N. Pham, T. Ibrahim, S.-i. Okihara, T. Furuhashi, T. Takasawa, K. Yasutomi, S. Kawahito, and H. Nagahara, "A dual-mode 303-megaframes-per-second charge-domain time-compressive computational cmos image sensor," *Sensors*, vol. 22, no. 5, p. 1953, 2022.

[31] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *ISCAS*, pp. 1–5, IEEE, 2021.

[32] R. Ikeno, K. Mori, M. Uno, K. Miyauchi, T. Isozaki, I. Takayanagi, J. Nakamura, S.-G. Wuu, L. Bainbridge, A. Berkovich, *et al.*, "A 4.6-μm, 127-db dynamic range, ultra-low power stacked digital pixel sensor with overlapped triple quantization," *T-ED*, vol. 69, no. 6, pp. 2943–2950, 2022.

[33] M.-W. Seo, M. Chu, H.-Y. Jung, S. Kim, J. Song, D. Bae, S. Lee, J. Lee, S.-Y. Kim, J. Lee, *et al.*, "2.45 e-rms low-random-noise, 598.5 mw low-power, and 1.2 kfps high-speed 2-mp global shutter cmos image sensor with pixel-level adc and memory," *JSSC*, vol. 57, no. 4, pp. 1125–1137, 2022.

[34] R. Goyal, S. Ebrahimi Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, *et al.*, "The" something something" video database for learning and evaluating visual common sense," in *ICCV*, pp. 5842–5850, 2017.

[35] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, *et al.*, "The kinetics human action video dataset," *arXiv preprint arXiv:1705.06950*, 2017.

[36] K. Soomro, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[37] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *ICCV*, pp. 4489–4497, 2015.

[38] B. Kellogg, V. Talla, S. Gollakota, and J. R. Smith, "Passive wi-fi: Bringing low power to wi-fi transmissions," in *NSDI*, pp. 151–164, 2016.

[39] "Nvidia reveals xavier soc details." https://www.forbes.com/sites/moorinsights/2018/08/24/nvidia-reveals-xavier-soc-details/amp/.

[40] G. K. Wallace, "The jpeg still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, 1991.

[41] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Deep convolutional autoencoder-based lossy image compression," in *PCS*, pp. 253–257, IEEE, 2018.

[42] T. Polonelli, D. Battistini, M. Rusci, D. Brunelli, and L. Benini, "An energy optimized jpeg encoder for parallel ultra-low-power processing-platforms," in *APPLEPIES 2019 7.*, pp. 125–133, Springer, 2020.

[43] J. Zhang, J. P. Newman, X. Wang, C. S. Thakur, J. Rattray, R. Etienne-Cummings, and M. A. Wilson, "A closed-loop, all-electronic pixel-wise adaptive imaging system for high dynamic range videography," *TCAS-I*, vol. 67, no. 6, pp. 1803–1814, 2020.

[44] T. Ma, W. Cao, F. Qiao, A. Chakrabarti, and X. Zhang, "Hogeye: neural approximation of hog feature extraction in rram-based 3d-stacked image sensors," in *ISLPED*, pp. 1–6, 2022.

[45] D.-W. Jee, S.-M. Ko, K. Kasichainula, I. Yeo, Y. Cao, and J.-S. Seo, "A time-memory-based cmos vision sensor with in-pixel temporal derivative computing for multi-mode image processing," in *ESSCIRC*, pp. 109–112, IEEE, 2023.

[46] R. LiKamWa, Y. Hou, Y. Gao, M. Polansky, and L. Zhong, "Redeye: Analog convnet image sensor architecture for continuous mobile vision," in *ISCA*, pp. 255–266, 2016.

Free and Fair Hardware: A Pathway to Copyright Infringement-Free Verilog Generation using LLMs

Sam Bush*, Matthew DeLorenzo*, Phat Tieu, Jeyavijayan Rajendran
Texas A&M University, USA
{samuelkbush, matthewdelorenzo, phat.tieu, jv.rajendran}@tamu.edu

***Abstract*—Limitations in Large Language Model (LLM) capabilities for hardware design tasks, such as generating functional Verilog codes, have motivated various fine-tuning optimizations utilizing curated hardware datasets from open-source repositories. However, these datasets remain limited in size and contain minimal checks on licensing for reuse, resulting in potential copyright violations by fine-tuned LLMs. Therefore, we propose an evaluation benchmark to estimate the risk of Verilog-trained LLMs to generate copyright-protected codes. To minimize this risk, we present an open-source Verilog dataset, `FreeSet`, containing over 220k files, along with the automated dataset curation framework utilized to provide additional guarantees of fair-use Verilog data. We then execute an LLM fine-tuning framework consisting of continual pre-training, resulting in a fine-tuned Llama model for Verilog, `FreeV`. Our results indicate that `FreeV` demonstrates the smallest risk of copyright-infringement among prior works, with only a 3% violation rate. Furthermore, experimental results demonstrate improvements in Verilog generation functionality over its baseline model, improving VerilogEval pass@10 rates by over 10%.**

***Index Terms*—Large Language Models, Copyright, Verilog**

I. Introduction

Recent advancements in the domain of generative artificial intelligence (AI) have resulted in the development of Large Language Models (LLMs) [1]. These models, consisting of billions of neurons composing deep neural network architectures [1], [2], have enabled the ability to effectively understand textual information and produce high-quality responses. These versatile capabilities, emerging from training procedures on vast amounts of text data, have led to their integration into a wide range of industries to automate workflows [3], specifically in code generation for software development [4]–[6]. Significant efforts have sought further optimizations of LLMs for code generation, largely through increases in the scale and quality of their training datasets [7], enhancing the syntax and functionality across a wide range of coding tasks [5]. These efforts, spearheaded by Microsoft [8], Meta [9], Google, and OpenAI [10], have resulted in widely adopted code-generation models and tools.

Taking inspiration from LLM performance in software, an emerging application of LLMs is in the chip design process [11]. This hardware design flow to develop integrated circuits (ICs) for use in electronic devices is becoming increasingly complex, error-prone, and expensive. Furthermore, an integral component of IC design and verification is enacted through code in hardware description languages (HDL), defining the behavior of digital circuit designs. This has motivated the utilization of LLMs in translating natural language hardware design requirements into register-transfer-level (RTL) code representations [12]–[14]. Research efforts have demonstrated optimizations in LLM for hardware design through a variety of approaches, including the inference process, fine-tuning the model itself on curated Verilog datasets [12], [15]–[17], reinforcement learning [18], and frameworks integrating prompt and tool feedback [13], [19], [20]. As a result, the ability of LLMs to produce compilable and syntactically correct Verilog has gradually improved.

Although these works demonstrate improvements to LLMs in the hardware domain, challenges remain within the fine-tuning process, limiting the opportunity for further LLM optimizations and practical adoption in hardware design. One is the availability of high-quality and real-world Verilog and RTL datasets for utilization in training, as compared to software languages, LLMs have a significantly smaller dataset corpus [12], [16]. Furthermore, prior efforts to curate Verilog datasets on open-source repositories have minimal filters for identifying licensing or copyright restrictions of these repositories, risking the unauthorized reuse of proprietary code by fine-tuned models. This is a pertinent concern for commercial LLMs, as OpenAI currently faces a lawsuit in which CoPilot is claimed to violate copyright law through open-source training procedures [21]. Furthermore, emerging threats of hardware design IP piracy [22] by LLMs can also be exacerbated through improper use of data in training and fine-tuning.

To overcome this challenge, we first (i) provide an open-source inference benchmark that evaluates the proclivity of Verilog-tuned LLMs to generate copyright-protected hardware designs based on a similarity threshold. (ii) We then propose an automated framework that enables the large-scale extraction of open-source Verilog files from Github, resulting in a curated high-quality Verilog dataset of over 220k diverse files, surmounting over 16GB of text data. Notably, unlike prior open-sourced datasets, this framework checks for permissive and non-permissive licenses on each repository along with file-by-file identification of copyright keywords to remove any codes with a high likelihood of copyright protection, minimizing the risk of LLM copyright infringement. (iii) This dataset is then utilized to perform continual pre-training on a state-of-the-art local LLM (`Llama-3.1-8B-Instruct`), enabling the

*These authors contributed equally to this work.

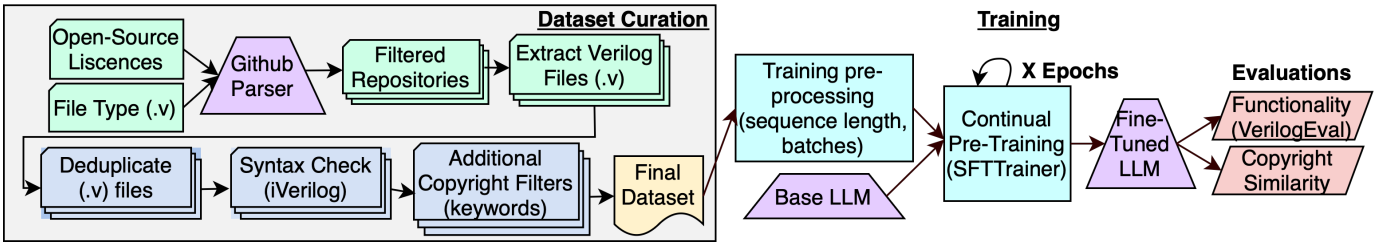


Fig. 1. Framework - Dataset Curation and Continual Pre-Training

functional performance and copyright similarity of the fine-tuned model to then be evaluated.

Our results demonstrate that FreeV has the smallest potential for hardware copyright infringement among prior works, with only a 3% violation rate. Furthermore, FreeV demonstrates improvements over its base Llama model when evaluated on VerilogEval, increasing the pass@5 rate by 7.9%. and the pass@10 rate by 10.1%.

Therefore in this paper, we provide the following contributions:

1) A novel benchmarking procedure in which risk of copyright infringement can be estimated in fine-tuned LLMs for hardware generation through similarity analysis.
2) To the best of our knowledge, we devised the first open-source Verilog dataset and curation framework for LLM fine-tuning that checks for both licenses and copyright content in each file, minimizing copyright risk.
3) A fine-tuned Llama-3.1-8B model on our Verilog dataset FreeV, demonstrating relative improvements in Verilog generation over its baseline and minimal hardware copyright violations.

II. Background and Related Work

A. LLM Risks in Copyright Infringement

As LLMs continue to scale in size and training datasets expand, the ability of LLMs to learn, memorize, and respond with texts that are copyright-protected is becoming an increasing risk to state-of-the-art models. Questions regarding the legal implications of the definition of "fair use" in model training and generation are actively debated [23], [24]. This has given rise to a series of lawsuits facing prominent generative-AI entities, including Stability AI in which plaintiffs allege improper use of their image content in training [25], along with claims by the New York Times that GPT-4 infringes copyright by reproducing articles in its output [26]. This has also impacted LLMs for code generation, in which Microsoft and OpenAI are claimed to have improperly utilized and monetized open-source programs [21]. These developments have motivated the need to develop frameworks to identify if copyrighted content in utilized in LLM training [27] and prevent the generation of copyrighted text [28].

This concern of improper use of code in training is relevant to the hardware domain regarding IP piracy as LLMs begin to be fine-tuned on hardware datasets. Beyond utilizing open-source code and preliminary license checks [17], minimal protections have been established in ensuring copyright-protected hardware is not utilized in LLM training.

B. LLM Fine-Tuning for Hardware Design

To effectively utilize emerging fine-tuning methodologies available for LLM optimization towards Verilog code generation tasks, a sufficiently sized and high-quality curated dataset is essential. Many works explore prompting state-of-the-art models (including GPT) for various RTL generation tasks [13], [14], [29], [30]. However, privacy limitations associated with API interfaces in commercial LLMs and limited optimization for specific use cases [31] motivate the use of open-source LLMs. Initial works such as VeriGen [12] first explored fine-tuning local models on a curated dataset of Verilog texts from GitHub and textbook data sources, demonstrating up to 41% improvement in Verilog code-generation tasks after additional fine-tuning on the dataset itself.

Furthermore, RTLCoder [16] proposed a methodology to leverage LLMs to supplement the Verilog dataset through LLM-assisted generation, in which Verilog codes are generated by GPT through a mutation-based framework. This work additionally generated English descriptions of the module, enabling instruction fine-tuning, in which LLMs are trained on ideal prompt and response structures for further LLM alignment. BetterV [17] proposes to augment the instruction-tuning Verilog dataset by pairing Verilog modules with C-translations, enabling the model's existing knowledge of C to be leveraged. CodeV [15] further expands Verilog datasets for instruction tuning by scraping GitHub for syntactically correct Verilog modules and prompting an LLM to generate effective summarizations of the code. Furthermore, [18] proposes the utilization of reinforcement learning through PPO, in which rewards are provided as feedback based on a golden hardware solution. To the best of our knowledge, only BetterV performs repository license filters in their curation process, with none indicating copyright checks on a file-by-file basis.

In addition to fine-tuning, researchers have proposed alternative methods of optimizing LLM performance in hardware design. This includes ChipChat [13] which demonstrates that through extended conversational prompting frameworks, larger scale hardware designs can be effectively generated. Feedback from tools such as human and EDA feedback are additionally

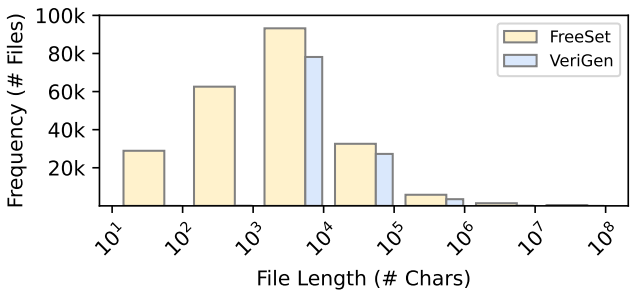


Fig. 2. Distribution in Verilog File Lengths of FreeSet vs. VeriGen [12]

proposed in [13], [14]. Optimizations to the inference (token-selection) process itself can be seen in [32], where Monte-Carlo Tree Search enables iterative feedback for PPA and functionality optimization. Additional guidance methodologies are proposed in [17] through discriminator guidance.

III. Framework

A. LLM Hardware Copyright Infringement Benchmark

We first examine the validity of the assumption that prior works training LLMs on Verilog datasets can risk copyright infringement through reproducing licensed code. Several LLMs have been trained using open-source code on GitHub and other code-sharing platforms [12], [15], [17]. Although they are trained on open-source material with some license checks, some files can contain copyrighted and proprietary materials.

To demonstrate this hypothesis, we employed prompt engineering with the intention to extract copyright training data from those LLMs. If these models generate copyrighted Verilog codes to a certain similarity threshold, the original training data for these models is likely to contain such copyrighted data. First, we curated a dataset with copyrighted materials that is available on GitHub. This dataset, collected in Section III-B, identifies copyright declarations in files often through comments and isolates the Verilog modules themselves for use in this benchmark. The result was a collection of 2K files with copyright licenses from several companies, such as Intel and Xilinx. Next, we used the text in these copyrighted materials as the input prompts for the LLMs. As these files still contained copyright-related information in the comments, all comments inside the files were stripped, leaving only the code behind. We then curated a set of 100 input prompts. Each prompt contained the first 20% of a copyrighted code file, with a limit of 64 words per prompt.

After feeding each prompt to the LLMs, the output was compared against the copyrighted dataset using cosine similarity. This score ranges from 0 to 1, where 0 represents no similarity, and 1 signifies an exact match. We use a threshold of 0.8 or greater to determine whether the output originated from the copyrighted dataset.

B. Dataset Curation

With the benchmark established to identify the risk of copyright infringement, we now seek to create a comprehensive dataset of varying file sizes that additionally ensures copyright-protected codes are not included. The framework (Figure. 1) for generating this dataset is described below.

1) ***Challenge 1. Extracting Verilog Data****:* One limitation is the ability to effectively extract significant amounts of Verilog files from online sources. Prior works have utilized tools including Google BigQuery, GitHub sources, and hardware design data extracted from textbooks, along with a series of augmentation techniques to expand a base set of data through translation [17] or mutations [16]. Furthermore, this work solely focuses on extracting and isolating high-quality data from real-world, open-sourced, and open-licensed sources, resulting in the utilization of data scraping tools. Furthermore, these APIs consist of their own challenges including costs, throughput bottlenecks from rate limitations, and granularity.

2) ***Solution 1****:* This challenge is addressed through the use of an automated system (implementing the GitHub API) to efficiently scrape repositories and extract Verilog code. This enables the dataset to including up-to-date Verilog code, as resources such as the Google BigQuery GitHub dataset used for VeriGen [12] have not been updated since 2022. At the time of curation, there are $\sim$50K publicly available GitHub repositories containing Verilog data, with a total of 1.3 million Verilog files. The GitHub API is limited to only being able to display 1K results per query for non-enterprise accounts. To avoid hitting this limitation the system was designed to granularize queries by repository creation date ranges, from 2008 when GitHub was established to 2024. Further granularization was performed by filtering for specific repository licenses, reducing the results per query significantly.

The resulting repositories in each query are then cloned to gather all of their data and author information for proper accreditation. Most files in the repositories were miscellaneous and consisted of non-Verilog data (i.e. readme files, binary test data, compiling information), so the Verilog data within the cloned repositories would then be filtered for and condensed into a large bank of Verilog files.

C. Removing Copyright-Protected Code

1) ***Challenge 2. Removing Copyright Protected Codes****:* An additional challenge using data gathered from GitHub is a lack of moderation regarding copyright restrictions. While GitHub enables massive sources of open-source data, many repositories lack proper licensing to ensure their data is not proprietary, and code bases under open-source licensing may still contain proprietary or copyrighted information. While most Verilog datasets from similar works [15], [16] that gather data from GitHub repositories report it to be open-source, the details and steps gone through to verify the data gathered are either non-existent or not comprehensive. This framework therefore implements additional precautions to ensure there is no data under private copyright within the training dataset.

2) ***Solution 2****:* As part of the process of granulating queries, a select set of commonly used open-source licenses, both permissive and non-permissive: MIT, Apache 2.0, variations

TABLE I
COMPARISON OF FREESET WITH PRIOR CURATED HARDWARE DESIGN DATASETS

LLM	Size (Disk)	Size (Rows)	Dataset Structure	Augmented (Y/N)	Open-Source	License Check
VeriGen's Dataset [12]	1.89 GB	108,971	Continual Pre-Training	No	Yes	No
RTLCoder [16]	55.1 MB	27,000	Instruction-Tuning	Yes	Yes	No
CodeV [15]	N/A	165,000	Instruction-Tuning	Yes	No	No
BetterV [17]	N/A	N/A	Instruction-Tuning	Yes	No	Yes
CraftRTL [33]	N/A	80,100	Instruction-Tuning	Yes	No	No
OriGen [34]	548 MB	222,075	Instruction-Tuning	Yes	Yes	No
FreeSet (This work)	**16.5 GB**	**222,624**	**Continual Pre-Training**	**No**	**Yes**	**Yes**

of the GNU General Public License and GNU Lesser GPL, Mozilla Public License, Creative Commons License, Eclipse Public Licenses, and BSD Licenses. While the intention of GitHub is for all repositories to be open-source, there are many repositories without a license on the repository that fall into a gray area in which they could potentially be part of a copyrighted code-base. As a result, these repositories were not considered for this dataset.

To ensure the dataset does not contain explicitly copyrighted data, a file-by-file filtering process is run to check the header comments of individual files for license information and combinations of language indicating the code to be under private copyright with combinations of keywords such as: "proprietary," "confidential," and "all rights reserved," ensuring that only repositories with the fair-use modules are utilized. In the de-duplicated version of the dataset (described in Section III-D1), this filter was able to locate over 2K files under proprietary copyright licenses from companies such as Intel, some even containing possible encryption keys and other critical information. As this data was contained within reportedly open-source code bases, all datasets that gathered data under the impression that none of the data on GitHub would be under private copyright would be at risk for including highly proprietary information.

D. Dataset Filtration

1) ***Challenge 3. Ensuring quality of dataset****:* When handling massive amounts of files, there are many potential issues and some considerations necessary to ensure quality. Often data, especially that of open-source files or recognized standards, will be reused heavily across thousands of files. This work will follow similar steps outlined in similar works (VeriGen [12]) for handling duplicated or reused data within the dataset. Another potential challenge arises from syntax or compiler errors within the training set code which can effectively train errors into a model trained on the set. Further, prior works have also found challenges with biased data in a training set containing data found in testing frameworks, this work follows similar methods to solve these issues.

2) ***Solution 3****:* The challenge of duplicated data within the set will bring the issue of over-fitting on potential models fine-tuned on the dataset, therefore de-duplication is required to clean the dataset for effective use. Following the method of de-duplication described in VeriGen, this work will implement a combined form of MinHash and Jaccard similarity. A method of Locality Sensitive Hashing (LSH) was implemented using MinHash signatures to represent each file. A similarity threshold of 0.85 was used to discard duplicate files, and LSH allowed for efficient querying for similar files in the dataset.

To additionally curate the dataset, the correctness of the syntax is then verified for each Verilog file as done in prior works [15], [17]. We implemented this through the use of *Icarus Verilog 10.3* to test the compilability of each code. As many files contain additional dependencies to modules outside of the files, only syntax-specific errors were identified and removed from the dataset in training.

E. Continual Pre-Training

1) ***Continual Pre-Training Setup****:* To perform continual pre-training using our dataset above, the SFTTrainer class from the Huggingface TRL library was utilized to implement the procedure in Python. We utilized the `Llama-3.1-8B-Instruct` as our base model for fine-tuning. The models were trained over 1 epoch across the full dataset, utilizing a single NVIDIA A100-40GB GPU. The training hyperparameters are defined as follows: max sequence length of 2048 tokens, and a training batch size per device of 16 with 2 gradient accumulation steps. Furthermore, to accommodate GPU memory limitations, QLoRA was applied to the model. This resulted in the quantization of the model's weights to 4-bit values and a LoRA layer with rank and alpha values equal to 8. Additionally, Unsloth was applied to accelerate the training process and minimize memory utilization.

2) ***Verilog Functionality Evaluation****:* To maintain consistency with the evaluations of prior works, we utilize the *VerilogEval-Human 1.0.0* benchmark to measure the Verilog functionality rates of our fine-tuned LLM. This benchmark consists of 156 Verilog generation problems with human-generated descriptions of the intended Verilog module. As done in prior works, the success of the LLM is defined by the pass@k metric below, specifying the proportion of Verilog modules that can be generated functionally correct at least once in k attempts from within a set of n total generations:

$$\text{pass@k} = \mathop{\mathbb{E}}_{\text{problems}} \left[\frac{1 - \binom{n-c}{k}}{\binom{n}{k}} \right] \tag{1}$$

The inferencing process was then performed on the base model `Llama-3.1-8B-Instruct` and our fine-tuned model,

TABLE II
COMPARISON OF LLMS CAPABILITIES ON VERILOGEVAL

Type	Model	Open-Source	Size	VerilogEval Human (%)		
				Pass@1	Pass@5	Pass@10
Foundation Models	GPT-4	No	N/A	43.5	55.8	58.9
	Codellama (CL)	Yes	7B	18.2	22.7	24.3
	DeepSeek-Coder (DS)	Yes	6.7B	30.2	33.9	34.9
	CodeQwen (CQ)	Yes	7B	22.5	26.1	28.0
Verilog-Tuned Models	VeriGen	Yes	16B	30.3	43.9	49.6
	RTLCoder-DS	Yes	7B	41.6	50.1	53.4
	BetterV-CodeQwen	No	7B	46.1	53.7	58.2
	CodeV-CodeQwen	Yes	7B	53.2	65.1	68.5
	OriGen-DS	Yes	7B	54.4	60.1	64.2
	CraftRTL-StarCoder2	No	15B	68.0	72.4	74.6
	OpenLLM-RTL	N/A	6.7B	42.8	51.6	55.0
This Work	Llama-3.1-Instruct (4-bit)	Yes	8B	14.8	23.0	25.9
	FreeV-Llama3.1 (4-bit)	Yes	8B	15.5	30.9	36.0

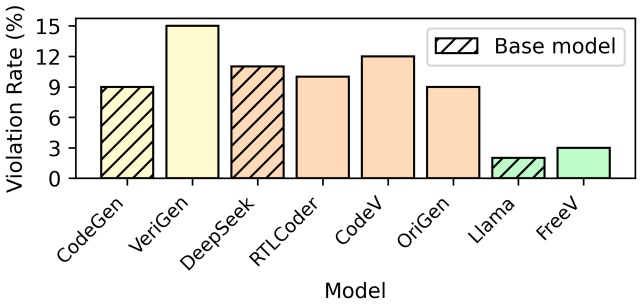


Fig. 3. Hardware Copyright Infringement Rates across LLMs

FreeV in their 4-bit quantized representations. This is due to memory limitations in our inferencing process, in which we utilize an NVIDIA A5000 24GB GPU. For both models, the temperature, or randomness of the token selection, was set to 0.2 and 0.8 and the best result was chosen to follow the precedence set by prior works. Furthermore, the inferencing process limits token generation to 2048 and is set to end at the first instance of "endmodule". The prompt for each VerilogEval definition simply consists of the VerilogEval-Human English description of the prompt, followed by the VerilogEval-Human prompt, consisting of the module instantiation, on the next line.

IV. EXPERIMENTAL EVALUATION

A. Evaluation of Dataset

The dataset configuration process described above gradually minimizes the dataset from its initial configuration. Through this minimization process, we can understand how each step impacts the dataset size. In the initial extraction of Verilog files from the repositories, over 1.3 million files were obtained. However, after extracting only items that have our set of open licenses, only 608,180 files remain. With this set of data, we can then perform LSH de-duplication which removed 62.5% of the files. Finally, syntax and copyright checks were performed, resulting in a final dataset of 222K files. Through this process, we can find that the copyrighted data made up nearly 1% of the original dataset. Through this minimization process, we can determine that certain steps such as de-duplication and copyright checking are crucial to minimizing the dataset.

With the dataset curated above, we can then compare its distribution of files in terms of character count to gain insight into the diversity of the dataset. This is shown in Figure 2, in which we provide a direct comparison against the open-source VeriGen dataset. Through this chart, we can see that our dataset contains a much higher frequency of smaller files, with the vast majority of files ranging from 10 to 10,000 characters. Furthermore, our dataset additionally contained some extreme outliers, including a file of over 90M characters. This diversity of data supports the dataset to be adaptable to a range of fine-tuning applications, from extended context length pre-training with models of large context windows (1M+ tokens) to smaller instruction-tuning pairs.

Additionally, we demonstrate additional dataset metrics in comparison to other open-source repositories, shown in Table I. Through these results, we see that our dataset is currently the largest open-source dataset in terms of size and number of Verilog files. This can be in part due to the curation procedures by other works such as CodeV, which removes files that exceed 2096 characters. Additionally, our dataset is notably the only open-source repository that contains explicit checks for licenses and copyright protection in each file (Table I).

B. Copyright Findings

In this section, we evaluate the results of the copyright detection framework defined in Section III-A. This evaluation (seen in Figure 3), was performed on our set of different LLMs to demonstrate the impact that fine-tuning LLMs on various hardware datasets has on the rate of hardware copyright violations (i.e., the proportion of tested copyright codes that exceed the similarity threshold). For each fine-tuned LLM, we also evaluate their associated base model, enabling the role of the fine-tuning process itself to be isolated in the induction of the violations, as information learned in pre-training of the underlying base model is a significant consideration in the generated results. The models used for the evaluation include `fine-tuned-codegen-6B-Verilog`

with the base model `codegen-6B-multi`, `RTLCoder-Deepseek-v1.1`, `OriGen` and `CodeV-DS-6.7B` with their base model `deepseek-coder-6.7b-base`, and our fine-tuned model with its base `Llama-3.1-8B-Instruct`.

Through these evaluations, we find that the fine-tuned models VeriGen and CodeV have a higher rate of copyright violations in comparison to their base models, with VeriGen increasing from 9% to 15%. This indicates that the fine-tuning process itself on the respective curated datasets plays a significant role in increasing the rates of generating copyright-protected hardware designs, with alternative methodologies and datasets potentially utilizing some of these copyright-protected codes in their fine-tuning dataset.

Furthermore, we can see that our fine-tuned model trained on hardware code without copyright protections, has the smallest hardware infringement rate (3%) out of all of the fine-tuned LLMs. Additionally, our model only depicts a 1% increase over its base model Llama-3.1. This further supports the notion that removing copyright-protected texts can significantly minimize the likelihood of generating these files in the inference process.

C. Evaluation of LLM Performance

After the continual pre-training is complete, the base `Llama-3.1-8B-Instruct` and our fine-tuned model `FreeV` are both evaluated on VerilogEval (Human) prompts through the methodology described in Section III-E2, and compared to the reported values from prior works [15], [33]. These include a set of foundational (pre-trained) models, along with state-of-the-art Verilog-Tuned models (Table II).

We can see that through continual pre-training on our curated dataset, `FreeV` demonstrates improvements in the ability to generate functional Verilog modules when compared to its base version. This is seen in improving the pass@1, pass@5, and pass@10 rates by 0.7%, 7.9%, and 10.1% respectively. However, we can also see that the current training process does not provide state-of-the-art performance in comparison to prior works, as `FreeV` does not outperform other fine-tuning models. This is largely expected, as most other works implement additional fine-tuning beyond causal pre-training, including instruction-tuning for further alignment. Also, there may be limitations due to 4-bit quantization.

However, we highlight that relative improvements are seen through pre-training on `FreeSet` when compared to the base `Llama-3.1-8B-instruct` model.

V. Discussion and Future Work

In this work, we seek to address the limitations of prior Verilog datasets with respect to copyright infringement of hardware designs. This is done through first establishing a preliminary evaluation benchmark that estimates how often fine-tuned LLMs generate a predefined set of copyright-protected hardware designs when prompted, with violations defined through a cosine-similarity threshold. Although this approach is feasible and easily implemented, future work can explore additional prompting methods to target certain texts the LLM may have learned through fine-tuning. Furthermore, the dataset of copyrighted texts can be further expanded tests through additional curation efforts to increase the robustness of the final determination. Lastly, other similarity metrics may be explored for effective comparisons of the hardware design, such as evaluating the design structure itself like GNN4IP [35].

The work then proposes an automated dataset curation framework, resulting in the largest open-source dataset of Verilog files, while additionally providing filters for copyright-protected codes. Future works may look to expand the current real-world texts through data-augmentation techniques, or through providing natural-language descriptions of the files for instruction-tuning purposes.

Lastly, we demonstrate an example fine-tuning procedure through continual pre-training on an base Llama model. Although this procedure successfully brought relative improvements over its baseline, future works may implement more extensive training parameters (epochs, batch size, larger LoRA adapters) to extract additional functionality. Furthermore, methodologies may adapt this dataset to their alternative training strategies (such as instruction-tuning or reinforcement learning), enabling their limitations to be tested.

VI. Conclusion

Large language models have shown promise in their application to the hardware domain field, specifically in the generation of RTL code. Recent works have demonstrated success in applying emerging LLM fine-tuning techniques to LLMs for the task of RTL generation, enabling state-of-the-art performance on RTL tasks. However, these works remain limited by the dataset size of the hardware designs used in training, along with minimal checks regarding copyright infringement requirements for the curated files, resulting in the risk of fine-tuned LLMs violating copyright protections.

To address this limitation, we first propose a preliminary benchmark formulation to estimate the risk that existing LLM trained on Verilog codes will generate copyright-protected texts to a certain similarity threshold, indicating the use of protected hardware designs in the dataset. In an effort to then minimize this risk, we propose an end-to-end automated framework for extracting large quantities of Verilog files from online sources like GitHub, while notably also filtering out any licensed or copyright-protected files. This framework results in a curated and open-source dataset `FreeV` for Verilog codes resulting in 16.5 GB and over 222k files. This dataset was then utilized to fine-tune an 8B parameter Llama model through continual pre-training for hardware code generation.

Our evaluations indicate that our model `FreeV` demonstrates the lowest frequency of copyright-infringement (only 3% of codes), while also providing Verilog functionality improvements over its base model, increasing VerilogEval pass@10 rates by over 10%.

Acknowledgment

The authors acknowledge the support from the Center for Secure Microelectronics Ecosystem – CSME#210205.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[2] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[3] Tim Keary, "12 Practical Large Language Model (LLM) Applications," https://www.techopedia.com/12-practical-large-language-model-llm-applications, 2023, [Online; last accessed 21-Nov-2023].

[4] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[5] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[6] Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Y. Li, H. Gao, S. Ma *et al.*, "Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence," *arXiv preprint arXiv:2406.11931*, 2024.

[7] D. Kocetkov, R. Li, L. B. Allal, J. Li, C. Mou, C. M. Ferrandis, Y. Jernite, M. Mitchell, S. Hughes, T. Wolf *et al.*, "The stack: 3 tb of permissively licensed source code," *arXiv preprint arXiv:2211.15533*, 2022.

[8] J. Stratton, "An introduction to microsoft copilot," in *Copilot for Microsoft 365: Harness the Power of Generative AI in the Microsoft Apps You Use Every Day*. Springer, 2024, pp. 19–35.

[9] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[10] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[11] R. Zhong, X. Du, S. Kai, Z. Tang, S. Xu, H.-L. Zhen, J. Hao, Q. Xu, M. Yuan, and J. Yan, "Llm4eda: Emerging progress in large language models for electronic design automation," *arXiv preprint arXiv:2401.12224*, 2023.

[12] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[13] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.

[14] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," *arXiv preprint arXiv:2305.14019*, 2023.

[15] Y. Zhao, D. Huang, C. Li, P. Jin, Z. Nan, T. Ma, L. Qi, Y. Pan, Z. Zhang, R. Zhang *et al.*, "Codev: Empowering llms for verilog generation through multi-level summarization," *arXiv preprint arXiv:2407.10424*.

[16] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[17] Z. Pei, H.-L. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," *arXiv preprint arXiv:2402.03375*, 2024.

[18] N. Wang, B. Yao, J. Zhou, X. Wang, Z. Jiang, and N. Guan, "Large language model for verilog generation with golden code feedback," *arXiv preprint arXiv:2407.18271*, 2024.

[19] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[20] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language model," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[21] B. Brittain, "Openai, microsoft want court to toss lawsuit accusing them of abusing open-source code," *Reuters*, Jan. 2023. [Online]. Available: https://www.reuters.com/legal/litigation/openai-microsoft-want-court-toss-lawsuit-accusing-them-abusing-open-source-code-2023-01-27/

[22] R. S. Chakraborty and S. Bhunia, "Rtl hardware ip protection using key-based control and data flow obfuscation," in *2010 23rd International Conference on VLSI Design*, 2010, pp. 405–410.

[23] M. Meeus, I. Shilov, M. Faysse, and Y.-A. de Montjoye, "Copyright traps for large language models," *arXiv preprint arXiv:2402.09363*, 2024.

[24] N. Rahman and E. Santacana, "Beyond fair use: Legal risk evaluation for training llms on copyrighted text," in *ICML Workshop on Generative AI and Law*, 2023.

[25] P. Samuelson, "Ongoing lawsuits could affect everyone who uses generative ai," *Science*, vol. 381, p. 6654, 2023.

[26] J. Freeman, C. Rippe, E. Debenedetti, and M. Andriushchenko, "Exploring memorization and copyright violation in frontier llms: A study of the new york times v. openai 2023 lawsuit," in *Neurips Safe Generative AI Workshop 2024*.

[27] H. Li, G. Deng, Y. Liu, K. Wang, Y. Li, T. Zhang, Y. Liu, G. Xu, G. Xu, and H. Wang, "Digger: Detecting copyright content mis-usage in large language model training," *arXiv preprint arXiv:2401.00676*, 2024.

[28] X. Liu, T. Sun, T. Xu, F. Wu, C. Wang, X. Wang, and J. Gao, "Shield: Evaluation and defense strategies for copyright compliance in llm text generation," *arXiv preprint arXiv:2406.12975*, 2024.

[29] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 722–727.

[30] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," *arXiv preprint arXiv:2306.14027*, 2023.

[31] S. Liu, Y. Lu, W. Fang, M. Li, and Z. Xie, "Openllm-rtl: Open dataset and benchmark for llm-aided design rtl generation," 2024.

[32] M. DeLorenzo, A. B. Chowdhury, V. Gohil, S. Thakur, R. Karri, S. Garg, and J. Rajendran, "Make every move count: Llm-based high-quality rtl code generation using mcts," *arXiv preprint arXiv:2402.03289*, 2024.

[33] M. Liu, Y.-D. Tsai, W. Zhou, and H. Ren, "Craftrtl: High-quality synthetic data generation for verilog code models with correct-by-construction non-textual representations and targeted code repair," *arXiv preprint arXiv:2409.12993*, 2024.

[34] F. Cui, C. Yin, K. Zhou, Y. Xiao, G. Sun, Q. Xu, Q. Guo, D. Song, D. Lin, X. Zhang *et al.*, "Origen: Enhancing rtl code generation with code-to-code augmentation and self-reflection," *arXiv preprint arXiv:2407.16237*, 2024.

[35] R. Yasaei, S.-Y. Yu, E. K. Naeini, and M. A. Al Faruque, "Gnn4ip: Graph neural network for hardware intellectual property piracy detection," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 217–222.

Asymmetric Predictive Testing for Aging in SRAMs

Yunkun Lin
Department of Electrical Engineering
University of Southern California
yunkunli@usc.edu

Mingye Li
Department of Electrical Engineering
University of Southern California
mingyel@usc.edu

Sandeep Gupta
Department of Electrical Engineering
University of Southern California
sandeep@usc.edu

***Abstract*—To avoid corruption of user data, predictive testing methods have been proposed to identify SRAMs likely to fail in the near future due to aging. These methods use aggressive operating conditions, e.g., adjustments to wordline voltages or the power supply voltage, that are calibrated to provide high coverage of SRAMs likely to fail in the near future, but end up with some over-testing, i.e., spuriously identifying some chips as likely to fail.**

We first present our study which discovered that a large fraction of over-tested chips fail due to read faults triggered during read-1 operations. Our analysis identifies asymmetric aging in SRAM cells, which are more likely to store zeros, as the root cause for this. We build on this discovery to propose an *asymmetric predictive testing method* which performs writes using normal voltages, read-0 at aggressive voltages, and read-1 at less aggressive voltages. We demonstrate that this method significantly reduces over-testing by over $3\times$ to $5\times$, for low limits on under-testing. We also propose and use a new statistical sampling and simulation method to enable fast convergence and accurate evaluation of asymmetric predictive testing.

***Index Terms*—Predictive tests, asymmetric predictive tests, SRAMs, process variation, over-testing, under-testing, aging, BTI, importance sampling, Monte Carlo simulations**

I. Introduction

Intel has reported an aging-induced V_{min} shift instability issue in its 13th and 14th Gen Core desktop processors [1]. Similarly, Google researchers in [2] highlighted significantly higher-than-expected failure rates in data center processors, identifying aging as a key factor contributing to these failures.

Static Random Access Memories (SRAMs) dominate processor chip areas. Hence, process variations and aging significantly influence SRAM performance, energy efficiency, and reliability. Process variations have become increasingly significant as feature sizes have decreased. Also, SRAM cells have become more susceptible to progressive degradation over their lifetimes due to aging mechanisms, primarily Bias Temperature Instability (BTI). (Hot Carrier Injection (HCI) is important for logic but less important for SRAMs.) These effects shift threshold voltages, slow down transistors, reduce noise margins, and can lead to failures. The cumulative effects of process variations and aging can result in a dramatic reduction in SRAM performance and reliability.

Prior research: Improving performance and resilience in the presence of process variations and aging effects has been an active study area [3]–[19]. Approaches that integrate sensors for real-time monitoring of SRAMs and adjusting their operating modes are outlined in [15]–[19]. Design modifications aimed at improving aging resilience are discussed in [9], [20], while techniques for managing operational modes, including periodic recovery phases, appear in [21], [22]. Post-manufacturing predictive tests to filter out chips prone to aging-induced failure are proposed in [3]–[8].

This research was supported by the Semiconductor Research Corporation (Task: 3176.001).

In this paper, we analyze the data regarding existing predictive tests, especially *under-testing,* i.e., the failure to reject chips that will fail due to future aging thereby potentially corrupting user data, and *over-testing,* i.e., the unnecessary rejection of chips that will remain functional despite future aging. We discover that a very large proportion of over-testing occurs due to errors provoked by read-1 operations. We further identify that this is rooted in the distribution of data values stored — zeros stored much more often than ones [23] — which causes asymmetric aging.

Based on this discovery, we propose a new asymmetric predictive testing method to further optimize the balance between over-testing and under-testing. This new method addresses the combined effects of process variations and aging and significantly reduces over-testing compared to prior predictive testing approaches, while ensuring that under-testing remains below a user-specified limit. Importantly, this new method achieves this without requiring complex SRAM redesign or active management of operational modes required by above prior research.

The evaluation of the effectiveness of our new predictive tests necessitates a new statistical sampling approach. We develop enhanced methods for importance sampling (IS) Monte Carlo (MC) simulations to accurately estimate yield, over-testing, and under-testing. This advances previous IS-MC methods by incorporating models of SRAM failures. We use this new IS-MC method to demonstrate that, for the same under-testing constraints, the asymmetric approach significantly reduces over-testing, compared to previous predictive testing methods [3]–[8]. We also demonstrate that the new IS-MC method achieves faster convergence and greater accuracy.

In Section II, we analyze the primary causes of over-testing and introduce our asymmetric predictive testing method. Section III summarizes the IS-MC simulation approach and demonstrates the effectiveness of our proposed predictive testing and simulation techniques. Finally, we conclude with a summary and outline directions for future research.

II. Predictive tests to mitigate aging in SRAMs

To address the aging problem in SRAMs, which worsens over time, our goal is to develop predictive tests for aging-induced faults to identify SRAMs that will fail with the level of aging expected with usage in the near future, i.e., over the next m months. The goal is to use the test results to take preemptive actions that help avoid corruption of user data due to aging failures.

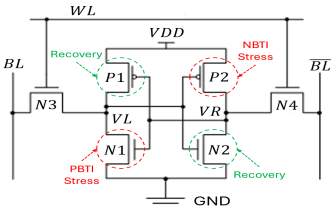


Fig. 1. 6T SRAM cell and the stress/recovery status when "0" is stored [24]

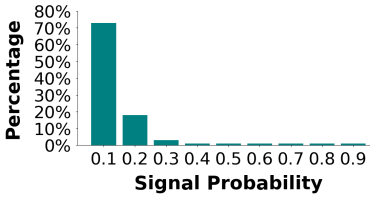


Fig. 2. The signal probability distribution of SRAM cells extracted from [23]

A. Background on SRAMs

(a) In this paper, we use as a case study a 32KByte SRAM array along with the 6T SRAM cell design, un-correlated Gaussian process variation model, a BTI aging model, post manufacturing yield, and aging induced failure rate (under-testing) computation methods from [3], [4], [25]–[27]. Our selection of this 32KByte SRAM array is motivated by its practicality. On one hand this size is widely used in low-level cache implementations, making it realistic as well as sufficiently large to capture all key features that affect predictive testing under the combined effects of process variation and aging. On the other hand, it is small enough to permit extensive simulation studies required to discover the key underlying trends and to thoroughly evaluate new methods.

(b) For estimating BTI aging, we use a real workload where the signal probability distribution in data caches is extracted [23]. This distribution is shown in Figure 2, where the signal probability is the probability that an SRAM cell stores the value "1" over its lifetime.

This distributions shows that SRAM cells that store data in processors are significantly more likely to store "0". When the 6T cell holds a "0", as illustrated in Figure 1, the N1 and P2 transistors experience stress while the N2 and P1 transistors undergo recovery. Consequently, a higher likelihood of the cell being in the "0" state results in a greater $|\Delta V_{th}|$ due to aging for the N1 and P2 transistors compared to N2 and P1.

(c) For fault model, we focus on *aging-induced* ***read delay faults*** [28] and ***read destructive faults*** [29]. A read delay fault occurs where there is insufficient time for bit or bit-bar line to discharge before the output of sense amplifier is outputted. A read destructive fault occurs when the value stored in a cell is flipped during a read operation.

B. Background on predictive testing for aging in SRAMs

Several predictive testing methods targeting read delay and destructive faults have been proposed in [3]–[8]. A primary criterion for selecting the most effective approaches is the balance they achieve between under-testing and over-testing to predict the aging failures in the future. Figure 3 illustrates this important guiding principle. Here, under-testing represents SRAMs that pass our predictive tests but fail after aging, and over-testing represents SRAMs that fail our predictive tests but still work after actual future aging.

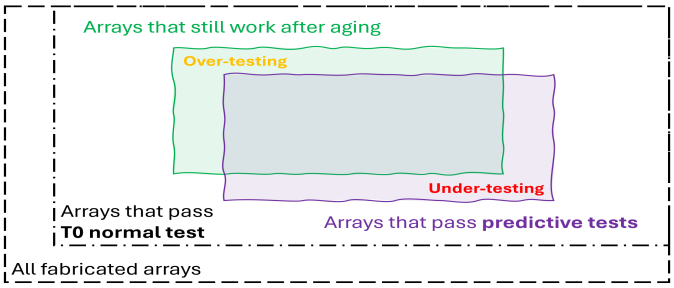


Fig. 3. A Venn diagram explaining the correlation between the post manufacturing (T0) normal test, predictive test, arrays that work after aging

Our ideal goal is to design predictive tests that achieve zero under-testing ***and*** zero over-testing. However, such a precise control during predictive testing is not possible since we only adjust one or two key voltages and/or frequencies during predictive testing. Hence, our practical objective is the following constrained-optimization.

Objective: Minimize total over-testing.

Constraint: Total under-testing $\leq \alpha$.

The minimization of over-testing maximizes a chip vendor's revenue while the constraint on under-testing is determined by the part-per-million (ppm) failure rates promised to customers by the vendor. For chip users, the maximum value of total under-testing, α, determines the probability of failures. In turn, this determines the probability of corruption of their data.

In our prior work [3] [4], the trade-offs between over-testing and under-testing are discussed for the **wordline over-drive tests** for predicting read destructive faults and **wordline under-drive tests** for predicting read delay faults.

We start this research by studying the above predictive tests using our improved IS-MC method, presented in Section III. To gain deeper insights into the root causes of over- and under-testing, we partitioned the data into finer categories. The most informative results are summarized in Table I for predicting read destructive faults and Table II for read delay faults.

Both these tables separately show under-testing and over-testing that occur during read-0 and read-1 operations for the prediction of 1-month aging under the skewed workload [23]. These tables do not show the counts for under-testing and over-testing of both read-0 and read-1 failures occurring in the same cell as these values are negligible. These tables show how over-testing varies with different wordline voltage adjustments. Our goal is to identify the root cause of over-testing in predictive testing and explore ways to mitigate.

C. Root cause of large over-testing given under-testing constrain for predictive testing - Read destructive faults

We start by studying the aging induced read destructive faults predicted by wordline over-drive tests, where a higher wordline voltage V_{WL} is applied, such that $V_{WL} = V_{WL,nominal} + \Delta V_{WL}$ during testing. This provides the following two observations.

Observation-1: As shown in Rows 2-4 of Table I, read-0 destructive faults after aging dominate the under-testing of read destructive faults tested by wordline over-drive, accounting for nearly all of the 3.48% aging-induced failure rate when predictive tests are not applied (see column 2 rows 2-4).

TABLE I
WORDLINE OVER-DRIVE [3] [4] PREDICTING **READ DESTRUCTIVE FAULTS**: OVER-TESTING AND UNDER-TESTING OF READ-0 AND READ-1 FOR **1-MONTH AGING** UNDER THE SKEWED WORKLOAD [23], FOR VARIOUS WORDLINE OVERDRIVE VOLTAGES

Wordline over-drive (ΔV_{WL}) value	0mV	2mV	4mV	6mV	8mV	10mV	12mV	14mV	16mV	18mV	20mV	21mV	22mV	23mV	24mV	25mV	26mV
Total Under-testing	3.48%	2.87%	2.57%	2.25%	1.95%	1.70%	1.41%	1.21%	0.90%	0.70%	0.51%	0.43%	0.33%	0.25%	0.19%	0.13%	0.08%
Read-0 only under-testing	3.48%	2.87%	2.57%	2.25%	1.95%	1.70%	1.41%	1.21%	0.90%	0.70%	0.51%	0.43%	0.33%	0.25%	0.19%	0.13%	0.08%
Read-1 only under-testing	≈ 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total Over-testing	0	0.06%	0.14%	0.22%	0.31%	0.41%	0.55%	0.67%	0.86%	1.07%	1.29%	1.35%	1.54%	1.62%	1.79%	1.92%	2.12%
Read-0 only over-testing	0	≈ 0	≈ 0	≈ 0	0.01%	0.03%	0.06%	0.08%	0.15%	0.20%	0.31%	0.35%	0.42%	0.48%	0.53%	0.59%	0.67%
Read-1 only over-testing	0	0.06%	0.14%	0.22%	0.30%	0.38%	0.49%	0.59%	0.71%	0.87%	0.98%	1.05%	1.12%	1.17%	1.26%	1.33%	1.45%

TABLE II
WORDLINE UNDER-DRIVE [3] PREDICTING **READ DELAY FAULTS**: OVER-TESTING AND UNDER-TESTING OF READ-0 AND READ-1 FOR **1-MONTH AGING** UNDER THE SKEWED WORKLOAD [23], FOR VARIOUS WORDLINE UNDERDRIVE VOLTAGES

Wordline under-drive ($-\Delta V_{WL}$) value	0mV	1mV	1.5mV	2mV	2.5mV	3mV	3.5mV
Total Under-testing	1.87%	1.11%	0.63%	0.30%	0.15%	0.06%	0.02%
Read-0 only under-testing	1.17%	0.79%	0.54%	0.27%	0.15%	0.06%	0.02%
Read-1 only under-testing	0.70%	0.32%	0.09%	0.03%	≈ 0	0	0
Total Over-testing	0	≈ 0	0.05%	0.25%	0.60%	1.06%	1.55%
Read-0 only over-testing	0	0	≈ 0	0.03%	0.14%	0.33%	0.55%
Read-1 only over-testing	0	0	0.05%	0.22%	0.46%	0.72%	1.00%

Observation-2: As shown in rows 5–8 of Table I, read-1 related over-testing dominates the overall over-testing, for example, with an under-testing threshold of less than 0.2%, we need to select $\Delta WL = 24mV$ wordline over-drive, and the over-testing rate for read-0 faults is 0.53%, compared to 1.26% for read-1 faults.

The above observations from failure data for existing predictive testing caused us to investigate their root cause(s). This provided the hypothesis that the asymmetry in stored data leads to asymmetric aging effects, which in turn contribute to the observed distribution of over-testing and under-testing for aging predictive testing.

Analysis to validate this hypothesis: Table I models SRAM aging based on real-world SRAM usage, incorporating the skewed workload extracted from traces for actual programs [23]. As shown in Fig. 2, this distribution is dominated by stored '0', which causes PMOS P2 to suffer more from NBTI aging compared to PMOS P1, and NMOS N1 to suffer more from PBTI aging compared to NMOS N2 as shown in Figure 1 and explained in Section II-A.

Consider read destructive faults caused by such aging, as shown in Fig 1, where 6T SRAM cell stores 0 during a read operation. After the bitlines are precharged and the wordline is activated, the voltage division between N3 and N1 determines the voltage of V_L during the read operation. Due to aging, N1 is weakened by PBTI, resulting in a higher voltage at V_L. At the same time, the trip voltage V_{trip} (the minimum input voltage at which an inverter's output falls) of the inverter consisting of N2 and P2 decreases as the strength of P2 is weakened by NBTI. Such higher V_L and lower V_{trip} together make it easier to flip the cell during the read operation, which corrupts the stored value. Hence, the asymmetric aging increases the probability of exciting the read destructive faults during read-0 operations.

In contrast, a similar analysis when the cell stores a '1' shows that the above shifts in threshold voltages due to asymmetric aging also *increases the probability of read destructive faults during read-1 operations, but by a much smaller amount relative to the increase for read-0.*

As can be seen in the above tables, stricter stress conditions during predictive testing, namely higher ΔV_{WL}, increases the ability of the predictive tests to detect faults due to future aging. However, for the same duration of future aging, asymmetric aging increases the severity of aging faults excited by read-0 much more than faults excited by read-1. Hence, the level of over-drive voltage required to satisfy the given under-testing constraint for read-0 excited faults causes significant over-testing of read-1 excited faults.

Hence, the above shows that due to asymmetric aging, read-0 destructive faults dominate under-testing, consistent with *Observation-1*, and read-1 destructive faults dominate over-testing, consistent with *Observation-2*. Simply, by analyzing wordline over-drive over-testing results and conducting circuit analysis, we have identified the root cause of over-testing in predictive testing for aging-induced read destructive faults.

D. Root cause of large over-testing given under-testing constrain for predictive testing - Read delay faults

We carried out a similar study for aging-induced read delay faults as predicted by the wordline under-drive test, where a reduced wordline voltage V_{WL} is used. As shown in Table II, we observe similar trends as for read destructive faults: aged read-0 delay faults dominate under-testing, while read-1 delay faults dominate over-testing tested by wordline under-drive.

We undertake a similar circuit analysis as above. After aging, PBTI aging degrades the pull-down transistor N1 which slows the discharge rate for read-0, causing read-0 related delay faults. Similarly, aging in N2 slows the discharge rate for read-1. Due to the skewed workload, where '0' is stored more frequently, N1 experiences greater aging compared to N2. Consequently, read-0 delay faults are more severe than read-1 delay faults.

Read delay faults are different from read destructive faults in one way, the read delay faults are less severe compared to read destructive faults. This is evident in the magnitudes of under-testing when $\Delta V_{WL} = 0$ in Tables II and I, namely 1.87% and 3.48% respectively for read delay and read destructive faults. This difference arises because read delay faults are solely caused by PBTI aging in the two pull-down NMOS transistors, N1 and N2, whereas read destructive faults are caused by both PBTI aging in N1 and N2 and NBTI aging in the two pull-up PMOS transistors, P1 and P2.

E. Proposed asymmetric predictive testing method - Read destructive faults

Our goal is to refine predictive testing methods to reduce over-testing while ensuring that we still satisfy the constraints on under-testing.

TABLE III
MARCH TEST FOR **SYMMETRIC** WORDLINE OVER-DRIVE PREDICTING AGING-INDUCED READ DESTRUCTIVE FAULTS

Operation	$\Updownarrow w0$	$\Updownarrow r0$	$\Updownarrow w1$	$\Updownarrow r1$
$\mathbf{\Delta V_{WL}}$	0	$\Delta V_{WL,A}$	0	$\Delta V_{WL,A}$
Positive/Negative	0	Positive	0	Positive

TABLE IV
MARCH TEST FOR **ASYMMETRIC** WORDLINE OVER-DRIVE PREDICTING AGING-INDUCED READ DESTRUCTIVE FAULTS

Operation	$\Updownarrow w0$	$\Updownarrow r0$	$\Updownarrow w1$	$\Updownarrow r1$
$\mathbf{\Delta V_{WL}}$	0	$\Delta V_{WL,C}$	0	$\Delta V_{WL,D}$
Positive/Negative	0	Positive	0	Positive

Asymmetric predictive test: The existing predictive tests increase wordline over-drive voltage, ΔV_{WL}, until the under-testing, primarily of read-0 excited faults, falls below the required limit, α. However, by the time we reach the appropriate value of ΔV_{WL}, these tests cause high over-testing of read-1 excited faults. Table III illustrates such a test where the over-drive voltage $\Delta V_{WL,A}$ is used for read-0 as well as read-1.

Key idea-1: Use different over-drive voltages during read-0 and read-1 operations. In particular, use a value of ΔV_{WL}, for read-0 operations to ensure low under-testing of read-0 excited faults, while using a lower value of ΔV_{WL} for read-1 operations to reduce over-testing of read-1 excited faults.

Such an *asymmetric predictive test* is illustrated in Table IV using over-drive voltage $\Delta V_{WL,C}$ during read-0 operations and $\Delta V_{WL,D}$ during read-1 operations, where $\Delta V_{WL,C} > \Delta V_{WL,D}$. This approach allows us to accurately predict read-0 faults by using a sufficiently high over-drive voltage $\Delta V_{WL,C}$ during read-0, while avoiding the additional over-testing for read-1 faults by using lower over-drive voltage over-drive voltage $\Delta V_{WL,D}$ during read-1.

Avoiding over-testing at half-selected cells: The goal of asymmetric predictive testing is to avoid over-testing, especially of read-1 excited read destructive faults. The above approach reduces such over-testing by using a lower over-drive voltages, $\Delta V_{WL,D}$ during read-1 operations (r1 in Table IV).

However, most SRAM array include multiple words per row. For example, the SRAM we are using as a case-study contains 8 64-bit words per row. In any SRAM with two or more words per row, during each write or read operation on one word inherently triggers a read operation (without data output) on the remaining words in that row. The cells in these other words are referred to as half-selected cells, as illustrated in Figure 4. Specifically, when a write operation is performed on a word, every other cell in that row is half-selected and a read operation is performed at each such cell. Further, each such half-selected cell executes a read-0 or a read-1 operation, depending on whether it was storing a 0 or a 1 prior to the operation.

Key idea-2: To avoid over-testing of read-1 excited faults, in our asymmetric tests we use normal wordline voltage, i.e., $\Delta V_{WL} = 0$, during all write operations, as shown in Table IV.

The use of zero over-/under-drive during all write operations, avoids any over-testing of read-0 and read-1 excited faults in cells that are half-selected during any write operation.

Key idea-3: The final issue that we address is about the read operations applied at half-selected cells during read operations. Next, we show how our wordline over-drive tests in Table IV avoid any over-testing in half-selected cells **and** guarantee the detection of all aging-induced read destructive faults.

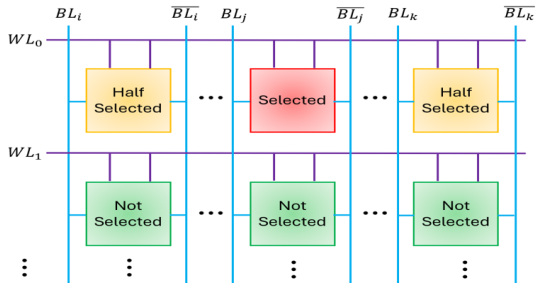


Fig. 4. Selected and half-selected SRAM cells during read/write operations

First, given the structure of the march test in Table IV, when a read-0 is performed on a cell, all the half-selected cells also contain '0' and hence the read operation performed on the half-selected cells is also read-0. Similarly, when a read-1 is performed on a cell, the operation performed at every half-selected cell is also read-1. Hence, all such half-selected reads are performed using the appropriate level of over-drive voltage and no additional over-testing occurs.

Second, when a stressed read operation is performed on a specific word, the wordline over-drive voltage may trigger an aging-induced read failure in a half-selected cell. In such a case, the erroneous value caused by the read destructive fault is stored in the corresponding cell, say C_{hs}. Further, since a single cell cannot exhibit both a read-0 and a read-1 destructive faults simultaneously (these fault types are mutually exclusive within distinct regions of V_{th}, as explained in Section III), our march test can safely continue with read operations on the remaining words in the same row without overwriting the above mentioned erroneous value capture in C_{hs}. Eventually, when we perform a read operation on cell C_{hs}, we read out the erroneous value stored when the read destructive fault was excited earlier, thereby detecting that fault.

In summary, our asymmetric predictive test method meets all our goals, and is summarized in Table IV. The first row of the table shows the operations to be applied sequentially to all words within each row and across all rows in the array. The last two rows specify the stress conditions during these operations for predictive testing. In our new asymmetric predictive testing, we use zero over-drive during all write operations and use distinct over-drive conditions for predictive testing during read-0 and read-1 operations, i.e., use $\Delta V_{WL,C}$ for read-0 and $\Delta V_{WL,D}$ for read-1 to enable asymmetric wordline over-drive testing, where $\Delta V_{WL,C} > \Delta V_{WL,D}$.

To determine the optimal stress conditions ($\Delta V_{WL,C}$ and $\Delta V_{WL,D}$) for predicting 1-month aged read destructive faults, we need to solve a constrained optimization problem. This requires ensuring that the total increase in under-testing for read-0 as well as read-1 related destructive faults remains below the threshold α, while minimizing the over-testing for both fault types. For under-testing below 0.2%, the optimal values are $\Delta V_{WL,C} = 24$ mV and $\Delta V_{WL,D} \approx 0$ mV, based on the over-testing and under-testing counts shown in Table I, in which the total under-testing is 0.19% while the over-testing is 0.53%. This over-testing is about $3.4\times$ smaller than the symmetric predictive testing which is 1.92%, which uses the

TABLE V
MARCH TEST FOR **SYMMETRIC** WORDLINE UNDER-DRIVE PREDICTING AGING-INDUCED READ DELAY FAULTS

Operation	$\Updownarrow w0$	$\Updownarrow r0$	$\Updownarrow w1$	$\Updownarrow r1$
$\mathbf{\Delta V_{WL}}$	0	$\Delta V_{WL,B}$	0	$\Delta V_{WL,B}$
Positive/Negative	0	Negative	0	Negative

TABLE VI
MARCH TEST FOR **ASYMMETRIC** WORDLINE UNDER-DRIVE PREDICTING AGING-INDUCED READ DELAY FAULTS

Operation	$\Updownarrow w0$	$\Updownarrow r0$	$\Updownarrow w1$	$\Updownarrow r1$
$\mathbf{\Delta V_{WL}}$	0	$\Delta V_{WL,E}$	0	$\Delta V_{WL,F}$
Positive/Negative	0	Negative	0	Negative

over-drive voltage $\Delta V_{WL,A} = 24$ mV for read-0 and read-1 operations, as shown in Table III.

F. Method to improve the over-testing issues for predictive testing - Read delay faults

As shown in Table V, existing predictive tests, such as wordline under-drive, use the same under-drive voltage $\Delta V_{WL,B}$ for both read-0 and read-1 operations.

Inspired by asymmetric wordline over-drive for read destructive faults, we apply different wordline under-drive voltages for read-0 and read-1 operations to predict read delay faults. Table VI outlines our approach: we use $\Delta V_{WL,E}$ for read-0 to reduce under-testing and $\Delta V_{WL,F}$ for read-1 to reduce over-testing. Both $\Delta V_{WL,E}$ and $\Delta V_{WL,F}$ are negative, with $|\Delta V_{WL,E}| > |\Delta V_{WL,F}|$. During all write operations, zero over-/under-drive is applied to prevent over-testing of read-0 and read-1 faults in half-selected cells.

Key Idea-4: We address the issue of read operations in half-selected cells triggering read delay faults. Table VI shows how our method prevents over-testing in these cells while ensuring the detection of all aging-induced read delay faults.

First, in the march test structure (Table VI), half-selected cells also perform read operations during a cell read. However, faults triggered in half-selected cells do not propagate to outputs, avoiding any over-testing. Second, if a read delay fault is triggered in a half-selected cell, C_{hs}, its stored value remains unchanged, ensuring no over-testing occurs when we perform a real read operation on that cell C_{hs}.

In summary, our asymmetric wordline under-drive method achieves all design goals and is detailed in Table VI, using a similar march test structure as the wordline over-drive method discussed in Section II-E.

Finally, to determine optimal stress conditions $\Delta V_{WL,E}$ for read-0 and $\Delta V_{WL,F}$ for read-1, we need to solve a constrained optimization problem minimizing over-testing while keeping under-testing within a set limit. For under-testing $< 0.2\%$, the optimal values are $\Delta V_{WL,E} = -2.7\text{mV}$ and $\Delta V_{WL,F} = -1.5\text{mV}$, resulting in total under-testing of 0.19% and over-testing of 0.2%, a $2.5\times$ reduction compared to symmetric predictive testing ($\sim 0.5\%$).

G. Method to improve the over-testing issues for predictive testing - Read destructive and delay faults

Building on the two approaches discussed above, our goal is to develop a predictive test method targeting read destructive and delay faults after aging, which reduces over-testing while satisfying the under-testing constraints.

TABLE VII
MARCH TESTS FOR **ASYMMETRIC** WORDLINE OVER- AND UNDER-DRIVE PREDICTING AGING-INDUCED READ DESTRUCTIVE AND DELAY FAULTS

Operation	$\Updownarrow w0$	$\Updownarrow r0$	$\Updownarrow r0$	$\Updownarrow w1$	$\Updownarrow r1$	$\Updownarrow r1$
$\mathbf{\Delta V_{WL}}$	0	$\Delta V_{WL,C}$	$\Delta V_{WL,E}$	0	$\Delta V_{WL,D}$	$\Delta V_{WL,F}$
Positive/Negative	0	Positive	Negative	0	Positive	Negative

Key idea-5: To reduce the predictive test's complexity, we combine the two march tests for asymmetric wordline under-drive and over-drive to form a new march test, in which the write operations are shared by these two test methods.

Because applying two different wordline voltages in the same read operation is not feasible, we apply wordline over-drive and under-drive during two separate read operations. As shown in Table VII, we begin by initializing the SRAMs with a write-0 operation under normal working conditions followed by a wordline over-drive read-0 with positive $\Delta V_{WL,C}$ to predict read-0 destructive faults, and then a wordline under-drive read-0 with negative $\Delta V_{WL,E}$ to predict read-0 delay faults. A similar procedure is followed for read-1 related faults. As we discussed in above, this method avoids extra over-testing and reduces test complexity from $8N$ to $6N$ compared to testing the two types of read faults separately.

To determine the optimal stressing conditions ($\Delta V_{WL,C}$, $\Delta V_{WL,D}$, $\Delta V_{WL,E}$, $\Delta V_{WL,F}$), we need to solve a constrained optimization problem aimed at minimizing total over-testing while keeping under-testing cases below a preset threshold (e.g., 0.2%) for predicting both read faults after aging. As discussed in Section III, the exclusivity of read delay and destructive faults, combined with the exclusive/independence of read-0 and read-1 destructive/delay faults, allows efficient computation of total counts. By summing the over-testing and under-testing values for read-0 and read-1 destructive and delay faults under the given stress conditions, the constrained optimization process is greatly simplified.

III. ENHANCED IMPORTANCE SAMPLING MONTE CARLO (IS-MC) SIMULATION METHOD AND RESULTS

Considering the fact that the failure rate, over-testing, and under-testing counts for a single cell are extremely small (10^{-8}), to address this, we apply IS-MC simulation methods to focus sampling on regions with higher failure likelihood, using weights to correct distribution bias. Due to page limit, we briefly outline our new IS-MC simulation approach.

A. Enhanced IS-MC simulation method in SRAM

Table VIII summarizes the dangerous regions for the six transistors corresponding to each failure mechanism, with '+' and '-' indicating stronger and weaker transistors, respectively, increasing read fault probabilities, and 'N/A' representing no impact. The table reveals that read-0 and read-1 destructive faults have distinct, mutually exclusive dangerous regions, with NMOS transistors N1 and N2 having opposing dangerous region. Similarly, read delay faults are independent, each with a unique dangerous region.

These exclusiveness and independencies create a concave failure rate relationship across the parameter space of the six transistors, rendering convex optimization methods from [30] [31] ineffective for SRAM simulations. To address this, we fit two independent sampling functions $g_{r0}(x)$ and $g_{r1}(x)$,

TABLE VIII
DANGEROUS REGIONS FOR ALL SIX TRANSISTORS IN A 6T SRAM CELL FOR READ DESTRUCTIVE AND DELAY FAULTS, '+' REPRESENTS STRONGER, '-' REPRESENTS WEAKER, 'N/A' REPRESENTS NO INFLUENCE

Fault type	Operation	N1	N2	N3	N4	P1	P2
Read Destructive	Read-0	-	+	+	N/A	N/A	-
	Read-1	+	-	N/A	+	-	N/A
Read Delay	Read-0	-	N/A	-	N/A	N/A	N/A
	Read-1	N/A	-	N/A	-	N/A	N/A

modeled as multidimensional Gaussian distributions for read-0 and read-1 faults' dangerous regions, respectively. Finally, The combined sampling function for IS-MC simulations is defined as: $\frac{1}{2}\left(g_{r0}(x) + g_{r1}(x)\right)$.

B. Experimental results for our asymmetric predictive testing

Using this new IS-MC simulation method, we analyze the impact of over-testing and under-testing by comparing results from our new asymmetric stress approach with the traditional symmetric stress method for read-0 and read-1.

Figures 5 and 6 illustrate the over-testing versus under-testing performance of two predictive testing methods for read destructive and delay faults over a 1-month aging period. The x-axis represents under-testing, and the y-axis represents over-testing. The red curve corresponds to the traditional symmetric stress method (Table III), while the green curve represents our proposed asymmetric stress approach.

As shown in the Figure 5, the new asymmetric stress method for read-0 and read-1 destructive faults performs better than the traditional symmetric approach by significantly reducing over-testing for identical under-testing constraints. For example, when we limit under-testing to below 0.2%, the asymmetric method achieves only 0.53% over-testing for a 1-month aging prediction, compared to 1.79% with the symmetric method. Similarly, as Figure 6 shows for read delay faults, the asymmetric method results in just 0.19% over-testing, whereas

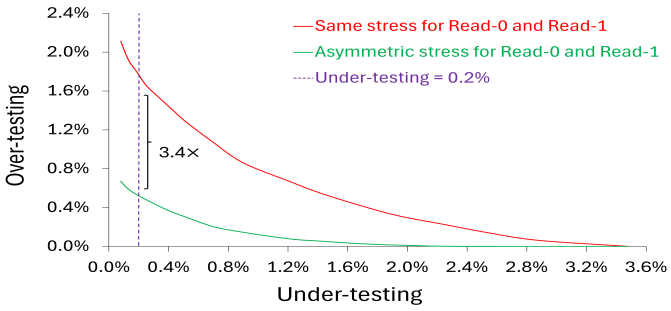


Fig. 5. Over-testing vs Under-testing for 1-month different predictive testing methods for read destructive faults based on methods in Table III and IV

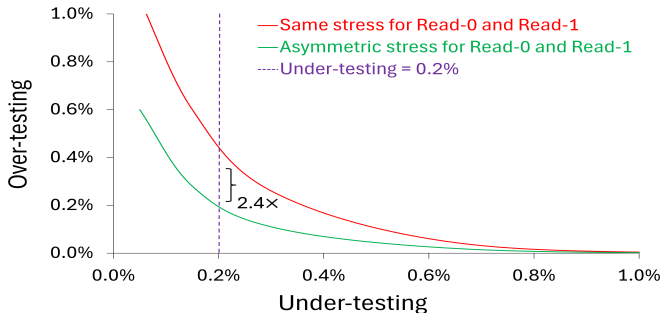


Fig. 6. Over-testing vs Under-testing for 1-month different predictive testing methods for read delay faults based on methods in Table V and VI

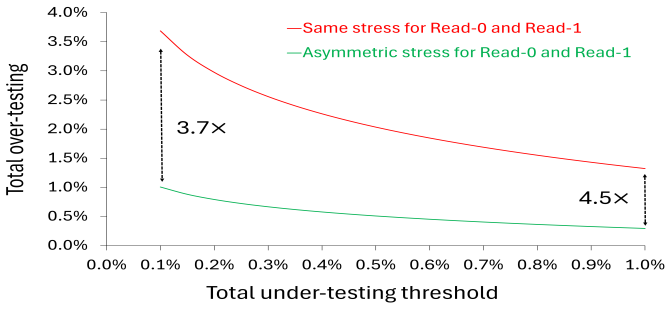


Fig. 7. Minimum over-testing for under-testing threshold ($0.1\% - 1.0\%$) for 1-month different predictive testing methods targeting both read destructive and delay faults based on method in Table VII

the symmetric approach has 0.46% over-testing. *These results demonstrate the superior efficiency of the asymmetric method for separately targeting read destructive and delay faults.*

Figure 7 compares asymmetric and symmetric predictive testing methods for read destructive and delay faults induced by 1-month aging. The x-axis represents the under-testing threshold (0.1% to 1%), while the y-axis shows the total over-testing count. As illustrated, the asymmetric predictive testing method achieves a $3\times$-$5\times$ reduction in over-testing at low under-testing thresholds, demonstrating its efficiency.

C. IS-MC simulation results

We analyzed the confidence ratio R_c for over-testing as a function of sample size, comparing our proposed IS-MC simulation method with the previous approach from [30] [31], where lower R_c values correspond to higher accuracy.

The **previous IS-MC method** consistently exhibits higher R_c values, indicating slower convergence and reduced precision. In contrast, the **enhanced IS-MC method** achieves significantly lower R_c values, reflecting faster convergence and greater accuracy. For example, at a sample size of 100,000, the enhanced method reaches an R_c of approximately 0.03, compared to about 0.15 for the previous method — $5\times$ higher. Notably, while the previous method struggles to meet the typical IS-MC convergence threshold of $R_c = 0.05$, our enhanced method comfortably satisfies this criterion.

These results highlight the significant advantages of the proposed IS-MC method in achieving faster convergence and greater accuracy, demonstrating its potential as an efficient improvement over traditional methods.

IV. CONCLUSION AND FUTURE RESEARCH

In this research, we present an enhanced predictive testing method designed to effectively mitigate over-testing when predicting failures caused by asymmetric aging effects. Using a proposed IS-MC simulation method, we demonstrate the efficiency and accuracy of our enhanced predictive testing approach in capturing aging-induced failure SRAMs.

Our ongoing work focuses on deriving the relationship between aging+ (process variation plus aging effect) and the fundamental trade-offs between over-testing and under-testing. Additionally, we are extending our methods by integrating other predictive testing techniques to further enhance testing efficiency for SRAMs and logic circuits.

REFERENCES

[1] Intel, "July 2024 update on instability reports on intel core 13th and 14th gen processors," *Intel Community Forum*, November 2024, accessed: 2024-11-14. [Online]. Available: https://community.intel.com/t5/Processors/July-2024-Update-on-Instability-Reports-on-Intel-Core-13th-and/m-p/1617113

[2] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2021, pp. 9–16.

[3] Y. Lin, L. Mingye, and G. Sandeep, "Predictive testing for aging in srams and mitigation," in *2024 IEEE International Test Conference (ITC)*. IEEE, 2024.

[4] J. Zhang and S. K. Gupta, "Wordline overdriving test: An effective predictive testing method for srams against bti aging," in *2017 18th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2017, pp. 454–459.

[5] J. J. Lee, A. Haggag, and W. Eklow, "Protecting against emerging vmin failures in advanced technology nodes," in *2014 International Test Conference*. IEEE, 2014, pp. 1–7.

[6] R. Kanj, R. Joshi, C. Adams, J. Warnock, and S. Nassif, "An elegant hardware-corroborated statistical repair and test methodology for conquering aging effects," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, 2009, pp. 497–504.

[7] B. Mohammad and P. Dadabhoy, "Effective screening for nbti effect on sram-based memory," in *2010 International Conference on Microelectronics*. IEEE, 2010, pp. 383–386.

[8] M. Ball, J. Rosal, R. McKee, W. Loh, T. Houston, R. Garcia, J. Raval, D. Li, R. Hollingsworth, R. Gury *et al.*, "A screening methodology for vmin drift in sram arrays with application to sub-65nm nodes," in *2006 International Electron Devices Meeting*. IEEE, 2006, pp. 1–4.

[9] X. Zuo and S. K. Gupta, "Asymmetric sizing: An effective design approach for sram cells against bti aging," in *2017 IEEE 35th VLSI Test Symposium (VTS)*, 2017, pp. 1–6.

[10] T. T.-H. Kim, W. Zhang, and C. H. Kim, "An sram reliability test macro for fully automated statistical measurements of v_{MIN} degradation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 3, pp. 584–593, 2011.

[11] A. Bansal, R. Rao, J.-J. Kim, S. Zafar, J. H. Stathis, and C.-T. Chuang, "Impact of nbti and pbti in sram bit-cells: Relative sensitivities and guidelines for application-specific target stability/performance," in *2009 IEEE International Reliability Physics Symposium*. IEEE, 2009, pp. 745–749.

[12] ——, "Impacts of nbti and pbti on sram static/dynamic noise margins and cell failure probability," *Microelectronics reliability*, vol. 49, no. 6, pp. 642–649, 2009.

[13] J. P. Kulkarni, K. Kim, and K. Roy, "A 160 mv robust schmitt trigger based subthreshold sram," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 10, pp. 2303–2313, 2007.

[14] V. Reddy, A. T. Krishnan, A. Marshall, J. Rodriguez, S. Natarajan, T. Rost, and S. Krishnan, "Impact of negative bias temperature instability on digital circuit reliability," *Microelectronics Reliability*, vol. 45, no. 1, pp. 31–38, 2005.

[15] P. Pouyan, E. Amat, and A. Rubio, "Adaptive proactive reconfiguration: A technique for process-variability-and aging-aware sram cache design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1951–1955, 2014.

[16] S. Duan, P. Wang, and G. Sai, "Bti aging monitoring based on sram start-up behavior," in *2020 IEEE 29th Asian Test Symposium (ATS)*. IEEE, 2020, pp. 1–6.

[17] Y. Aslan, F. Cacho, T. Kumar, D. Janardan, A. Kumar, F. Giner, M. Faurichon, and L. Anghel, "Minimum sram retention voltage: Insight about optimizing power efficiency across temperature profile, process variation and aging," in *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2023, pp. 1–6.

[18] S. Mukhopadhyay, K. Kim, H. Mahmoodi, A. Datta, D. Park, and K. Roy, "Self-repairing sram for reducing parametric failures in nanoscaled memory," in *2006 Symposium on VLSI Circuits, 2006. Digest of Technical Papers.*, 2006, pp. 132–133.

[19] S. Mukhopadhyay, K. Kang, H. Mahmoodi, and K. Roy, "Reliable and self-repairing sram in nano-scale technologies using leakage and delay monitoring," in *IEEE International Conference on Test, 2005.*, 2005, pp. 10 pp.–1135.

[20] J. Samandari-Rad and R. Hughey, "Power/energy minimization techniques for variability-aware high-performance 16-nm 6t-sram," *IEEE Access*, vol. 4, pp. 594–613, 2016.

[21] S. Ismail, S. Hossain, I. Hossain, and Y. Arafat, "Sram nbti mitigation with predictable activity based dynamic stress-recovery timing," in *2014 International Conference on Informatics, Electronics Vision (ICIEV)*, 2014, pp. 1–6.

[22] M. Yamaoka, K. Osada, and T. Kawahara, "A cell-activation-time controlled sram for low-voltage operation in dvfs socs using dynamic stability analysis," in *ESSCIRC 2008-34th European Solid-State Circuits Conference*. IEEE, 2008, pp. 286–289.

[23] C.-C. Chen, T. Liu, and L. Milor, "System-level modeling of microprocessor reliability degradation due to bias temperature instability and hot carrier injection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 8, pp. 2712–2725, 2016.

[24] X. Zuo, "Tdesign and testing of srams resilient to bias temperature instability (bti) aging," Ph.D. dissertation, UNIVERSITY OF SOUTHERN CALIFORNIA, Los Angeles, CA, May 2020.

[25] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, "Predictive modeling of the nbti effect for reliable design," in *IEEE Custom Integrated Circuits Conference 2006*. IEEE, 2006, pp. 189–192.

[26] J. Zhang and S. Gupta, "Sram array yield estimation under spatially-correlated process variation," in *2014 IEEE 23rd Asian Test Symposium*. IEEE, 2014, pp. 149–155.

[27] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Varius: A model of process variation and resulting timing errors for microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, no. 1, pp. 3–13, 2008.

[28] R.-F. Huang, Y.-T. Lai, Y.-F. Chou, and C.-W. Wu, "Sram delay fault modeling and test algorithm development," in *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, 2004, pp. 104–109.

[29] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, S. Borri, and M. Hage-Hassan, "Dynamic read destructive fault in embedded-srams: analysis and march test solution," in *Proceedings. Ninth IEEE European Test Symposium, 2004. ETS 2004.*, 2004, pp. 140–145.

[30] J. Wang, S. Yaldiz, X. Li, and L. T. Pileggi, "Sram parametric failure analysis," in *Proceedings of the 46th annual design automation conference*, 2009, pp. 496–501.

[31] J. Wang, X. Li, and L. T. Pileggi, "Parameterized macromodeling for analog system-level design exploration," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 940–943.

SmaRTLy: RTL Optimization with Logic Inferencing and Structural Rebuilding

Chengxi Li
Dept. of CSE
The Chinese University of Hong Kong
Hong Kong, China
cxli23@cse.cuhk.edu.hk

Yang Sun
Dept. of CSE
The Chinese University of Hong Kong
Hong Kong, China
ysun22@cse.cuhk.edu.hk

Lei Chen
Noah's Ark Lab
Huawei
Hong Kong, China
lc.leichen@huawei.com

Yiwen Wang
Noah's Ark Lab
Huawei
Hong Kong, China
wangyiwen19@huawei.com

Mingxuan Yuan
Noah's Ark Lab
Huawei
Hong Kong, China
yuan.mingxuan@huawei.com

Evangeline F.Y. Young
Dept. of CSE
The Chinese University of Hong Kong
Hong Kong, China
fyyoung@cse.cuhk.edu.hk

***Abstract*—This paper proposes smaRTLy: a new optimization technique for multiplexers in Register-Transfer Level (RTL) logic synthesis. Multiplexer trees are very common in RTL designs, and traditional tools like Yosys optimize them by traversing the tree and monitoring control port values. However, this method does not fully exploit the intrinsic logical relationships among signals or the potential for structural optimization. To address these limitations, we develop innovative strategies to remove redundant multiplexer trees and restructure the remaining ones, significantly reducing the overall gate count. We evaluate smaRTLy on the IWLS-2005 and RISC-V benchmarks, achieving an additional 8.95% reduction in AIG area compared to Yosys. We also evaluate smaRTLy on an industrial benchmark in the scale of millions of gates, results show that smaRTLy can remove 47.2% more AIG area than Yosys. These results demonstrate the effectiveness of our logic inferencing and structural rebuilding techniques in enhancing the RTL optimization process, leading to more efficient hardware designs.**

***Index Terms*—Multiplexer, Optimization, RTL, Synthesis, Structure**

I. Introduction

Logic optimization is an important part in logic synthesis. Multiplexers, often abbreviated as MUX, are common and important in digital circuits. In communication systems, they enable the transmission of multiple signals over a single communication line, thereby optimizing the use of bandwidth. In data routing, multiplexers facilitate efficient transfer of data between different parts of a computer system. Additionally, they are used in implementing Boolean functions and in the design of complex digital circuits[7, 10]. Therefore, optimization on multiplexers plays an important role in logic optimization[4, 5, 6, 12].

A common 2-to-1 MUX consists of two data input ports (A and B), a single-bit control port (S), and an output port (Y). More complex n-to-1 MUX can be formed by combining multiple 2-to-1 MUXs. Multiple MUX can form a tree in a combinational circuit to represent *if-else* and *case* statements in Hardware Description Language (HDL), which is called muxtree[8].

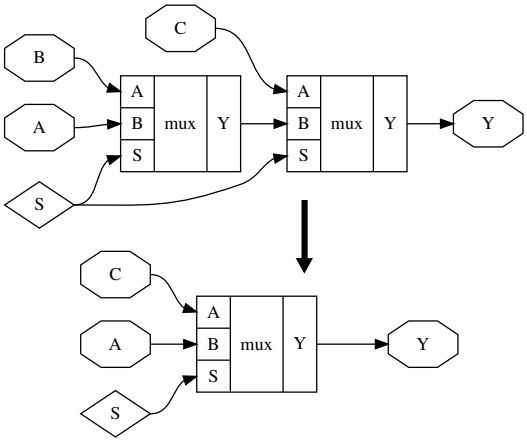


Fig. 1: The multiplexer with the same control as the ancestor multiplexer can be optimized.

Many logic synthesis tools include optimization pass on muxtree[3, 9, 13]. Yosys[15], a famous open-source Verilog HDL synthesis tool, primarily employs the *opt_muxtree* pass to optimize multiplexer trees. This pass analyzes control signals to identify and remove never-active branches by traversing the multiplexer trees and monitoring the values of visited control ports. A MUX will be removed if it shares the same control signal with visited MUXs. Specifically, it will optimize muxtree in the following two situations. First, when the value of the control port is determined in an ancestor MUX, as shown in Figure 1, the muxtree of $Y = S\ ?\ (S\ ?\ A : B) : C$ can be optimized into $Y = S\ ?\ A : C$. Second, when the value of the data port is determined in an ancestor MUX, as shown in Figure 2, the muxtree of $Y = S\ ?\ (A\ ?\ S : B) : C$ can be optimized into $Y = S\ ?\ (A\ ?\ 1 : B) : C$.

The method is feasible and efficient. In most cases, it can remove abundant gates and wires. However, the method has the following disadvantages:

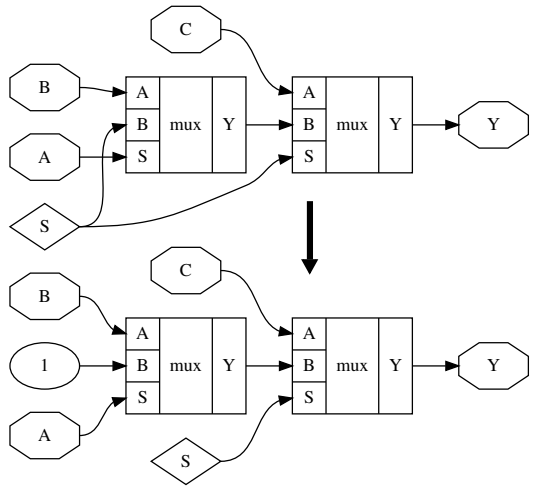


Fig. 2: The data port with the same control signals as the ancestor multiplexer can be optimized.

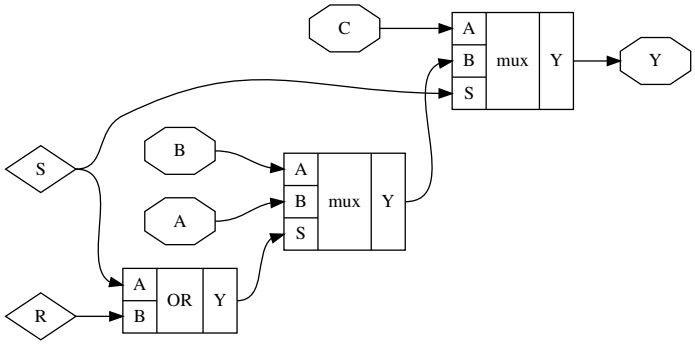


Fig. 3: The muxtree with different and logically dependent control signals.

- Limited logical analysis: The method ignores intrinsic logical relationships between signals. Traditional optimization occurs only when MUXs share identical signals, failing to optimize when control ports differ but are dependent. This limitation is illustrated in Figure 3, the muxtree of $Y = S\ ?\ ((S|R)\ ?\ A : B) : C$ can be optimized into $Y = S\ ?\ A : C$, because $S|R = 1$ when $S = 1$. However, this can be ignored in traditional methods.
- Structural inflexibility: Many approaches merely remove unused branches of a muxtree without changing its structure. However, many muxtrees can be optimized by rebuilding its structure for improved performance. A more detailed explanation will be given in section III.

To address these problems, we propose smaRTLy, which contains two optimizations, SAT-based redundancy elimination on sub-graphs and muxtree restructuring.

For SAT-based redundancy elimination, we focus on optimizing multiplexers with different control signals. Our approach involves analyzing the control port values and using an SAT solver to identify and eliminate redundant multiplexers. To manage complexity, we constrain the analysis to a sub-graph of the circuit and minimize the size of the sub-graph without sacrificing correctness.

For muxtree restructuring, we target at muxtrees with inefficient structures to eliminate more gates. We identify these structures, assess their efficiency, and use an Algebraic Decision Diagram (ADD) to rebuild the muxtree with fewer multiplexers [2]. This process involves reassigning control signals and outputs to optimize the overall design.

SmaRTLy were evaluated on 10 cases from IWLS-2005 [1] and RISC-V [14] benchmarks compatible with Yosys. Results show an additional 8.95% reduction in AIG areas compared to Yosys. The methods can be combined for enhanced performance and have the potential to integrate with other optimization techniques. Besides, smaRTLy can perform much better in an industrial benchmark, where the average size reaches millions of gates, and smaRTLy can remove 47.2% more AIG area than Yosys. This shows the effectiveness of smaRTLy on practical cases.

SmaRTLy offers improvements in circuit optimization, potentially leading to better PPA (Power, Performance, Area). As digital circuits grow more complex, such optimization techniques become increasingly valuable in managing design complexity and improving the overall system performance. Our contributions in this work are summarized below:

- We propose an SAT-based redundancy elimination to capture the logical relationship between signals.
- A novel method was developed that can greatly reduce the size of an RTL sub-graph.
- We propose an efficient muxtree restructuring algorithm using heuristics based on Algebraic Decision Diagram (ADD).
- Experimental results demonstrate improved optimization compared to traditional RTL optimizations in Yosys, highlighting the effectiveness of smaRTLy.

The rest of the paper is organized as follows. Section 2 presents SAT-based redundancy elimination. Section 3 presents Muxtree Restructuring, followed by experiment results in Section 4. Finally, Section 5 concludes the paper.

II. SAT-BASED REDUNDANCY ELIMINATION

The frequent use of *if-else* and *case* statements in RTL-level code often results in a significant number of multiplexers (MUXs) being generated during logical synthesis. This is a natural consequence of designing complex circuits, which typically involve multiple layers of nesting and numerous conditional statements. As a result, muxtrees become a prevalent feature in today's designs. While muxtrees are essential for implementing intricate logic, they can also lead to inefficiencies, particularly when they contain redundant MUXs. As the design logic becomes increasingly intricate, it is common to find redundant MUXs within these muxtrees.

A MUX is considered redundant if the value of its control port is already determined when traversing from the root to the MUX itself. In other words, the control signal's value can be deduced from known signals along the path. Similarly, optimization can be applied to the data port by analyzing these signals. Current tools like Yosys primarily optimize MUXs

with identical control signals. However, this approach overlooks many scenarios in digital circuits where control signals are interconnected through logic gates. For instance, complex logical relationships between control signals are common, as illustrated in Figure 3.

To address this limitation, we propose a SAT-based redundancy elimination method on the sub-graph to uncover deeper logical relationships. SAT (Boolean Satisfiability) solvers are powerful tools for determining whether there exists an assignment of variables that satisfies a given Boolean formula. By simplifying the sub-graph based on signal locations and applying SAT-based techniques on the sub-graph to the MUX optimization problem, it becomes possible to identify and eliminate redundancies that are not detectable through simpler methods like simulation.

SmaRTLy begins by constructing a sub-graph during the traversal of the muxtree. When a new MUX is encountered, all logical gates within a specified distance *k* from the control port are incorporated, provided they are not already included in the sub-graph. This approach allows for a more comprehensive analysis of the logical relationships between signals. However, if *k* is large, the size of the sub-graph will be too large for the SAT solver to take too much time; if *k* is small, the sub-graph will not contain enough nodes for the SAT solver to infer the value of the target. This is because the sub-graph contains too many irrelevant gates and signals, making the sub-graph bloated. To keep the sub-graph manageable, smaRTLy only adds potential signals whose values might be affected by known signals. We propose the scenarios in which the signal S can impact T:

Theorem II.1. *In the RTL level circuit, a signal S is possible to affect the value of signal T, only if:*

- *S is an ancestor of T, or*
- *T is an ancestor of S, or*
- *S and T have a common ancestor.*

The proof is not complicated; we only need to consider the scenarios not in Theorem II.1:

Proof. If the relationship between signal S and signal T is not in the three scenarios of Theorem II.1, then:

- There's no path from S to T (i.e. S and T are not connected), or
- S and T only have a common descendant.

In both scenarios, S is impossible to affect the value of T. □

The three scenarios are shown in Figure 4. We can find that T is also possible to affect S:

Theorem II.2. *If signal S is possible to affect the value of signal T, then T is also possible to affect the value of S.*

Proof. When S is possible to affect the value of T:

- If S is an ancestor of T, then T is a descendant of S.
- If T is an ancestor of S, then S is a descendant of T.
- If S and T have a common ancestor, then T and S have a common ancestor

So T is also possible to affect the value of S. □

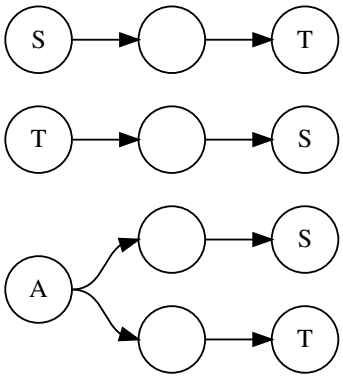


Fig. 4: The three situations where S(T) can affect T(S).

TABLE I: Inference rules for *OR* cells.

Condition	Result
$a = true$	$a \vert b = true$
$b = true$	$a \vert b = true$
$a = b = false$	$a \vert b = false$
$a \vert b = false$	$a = b = false$
$a \vert b = true\ a = false$	$b = true$
$a \vert b = true\ b = false$	$a = true$

From Theorem II.1 and Theorem II.2, we can see that these signals can be partitioned into groups such that signals in the same group will affect each other while those from different groups will not. This allows us to disregard gates without potential signals in effect, streamlining our analysis. The method can dismiss about 80% gates in the sub-graph, greatly accelerating the inference of the SAT solver. Additionally, smaRTLy excludes sequential gates to maintain the sub-graph as a Directed Acyclic Graph (DAG).

To enhance the efficiency of the SAT solver, smaRTLy then employs inference rules. Considering that the logical relationships are often not overly complex, as illustrated in Figure 3, straightforward inferences can help reduce unknown signals. smaRTLy apply the inference rules shown in Table I to the known value signals. If a condition matches, the corresponding signal in the result becomes a new known value signal.

Lastly, smaRTLy performs both simulation and SAT solver techniques on the sub-graph to determine signal values. To optimize efficiency, we choose between these methods based on the number of inputs requiring assignment. For a smaller number of inputs, simulation is more efficient, while the SAT solver is better suited for handling larger sets of inputs. The determination of a signal S is achieved by checking if either $SAT(S = 0) = false$ or $SAT(S = 1) = false$, indicating that the signal's value is fixed. In smaRTLy, we employ MiniSAT [11], a robust and efficient SAT solver, to handle complex logical evaluations. While there are more recent

```
case(S)
    2'b00:   Y = p0;
    2'b01:   Y = p1;
    2'b10:   Y = p2;
    default: Y = p3;
endcase
```

Listing 1: HDL code for a case statement

SAT solvers that might offer improved performance in certain scenarios, MiniSAT remains a popular choice due to its balance of efficiency and ease of integration.

To prevent excessive computational overhead, we introduce a threshold for the number of inputs. If the number of inputs in the sub-graph becomes excessively large, they may choose to forgo the SAT process. This decision is made to prevent the optimization process from becoming a bottleneck in the overall circuit synthesis workflow. This selective approach ensures that our method remains practical and responsive to varying complexities within the sub-graph.

III. Muxtree Restructuring

Yosys focuses on removing redundant MUXs from muxtrees without changing their structure and control ports. However, reconstructing these muxtrees into more efficient structures makes it possible to reduce the number of MUXs and connected gates further. This structural change can minimize resource usage and decrease circuit area and delay, ultimately enhancing the circuit's performance and sustainability.

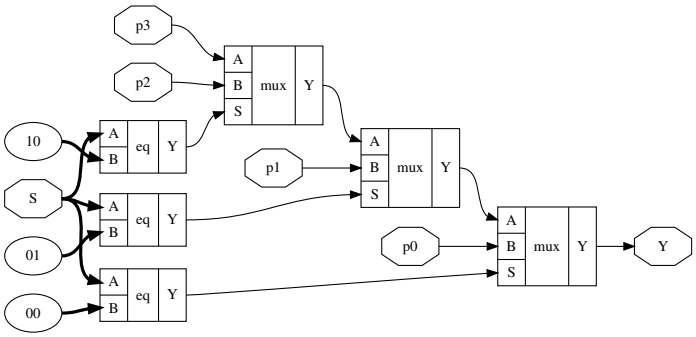


Fig. 5: A muxtree for the *case* statement in Listing 1. The muxtree is a chain.

In general, smaRTLy will rebuild muxtrees with a single control signal into a better structure. For example, the HDL code in Listing 1 is represented as a muxtree with three *eq* gates and three MUX gates, as illustrated in Figure 5 and Figure 6. The muxtree in Figure 5 forms a chain, which is inherently inefficient due to its sequential nature. In contrast, the muxtree in Figure 6 is designed as a full binary tree with an additional *or* gate. While this structure improves efficiency, it fails to account for the logical relationships between control ports. Their control signals are all signal S, so it can be formed as a more efficient structure By revising the structure and control ports, the muxtree can be optimized to utilize only

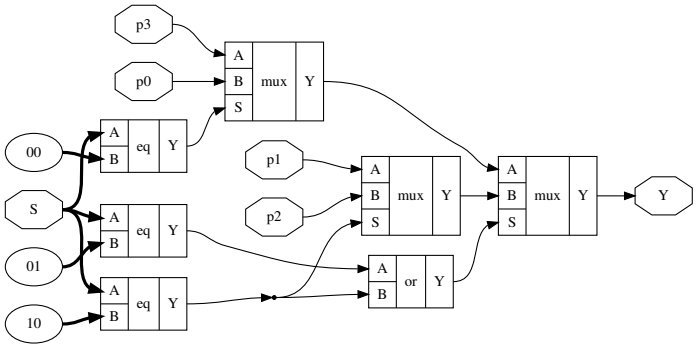


Fig. 6: Another muxtree for the *case* statement in Listing 1. The muxtree is a full binary tree.

three MUX gates, as demonstrated in Figure 7. This approach not only simplifies the design but also enhances its efficiency by eliminating redundant components. The three *eq* gates are disconnected from the muxtree and will be removed if they do not connect to any other gates, further simplifying the circuit.

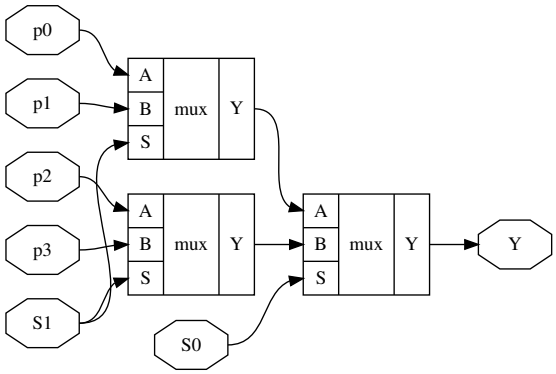


Fig. 7: A rebuilt muxtree connected with fewer gates.

Algorithm 1 Muxtree Restructuring

```
for cell in {muxtree_roots} do
    if OnlyEq(cell) and SingleCtrl(cell) then
        Assignment ← ADD(cell)
        RemovedEq ← CountRemoved(cell)
        height ← cell.ctrl.width
        width ← cell.width
        if Check(Assignment, RemovedEq, height, width) then
            Rebuild(cell, Assignment)
            RemoveUnusedCell()        ▷ Implemented in other pass
        end if
    end if
end for
```

The algorithm of Muxtree Restructuring is shown in Algorithm 1. smaRTLy first identifies all muxtrees with structures that can be optimized. In this paper, we focus on muxtrees generated from *case* statements, as they typically use *eq* gates

```
case(S)
    3'b1zz:   Y = p0;
    3'b01z:   Y = p1;
    3'b001:   Y = p2;
    default:  Y = p3;
endcase
```

Listing 2: HDL code for another case statement

or *logic_not* as control ports (*logic_not* is a special *eq*) and share the same control signal.

Next, smaRTLy assigns signals to the control ports of each MUX. A well-chosen assignment enables us to rebuild the muxtree with fewer MUXs. Conversely, a poor assignment can lead to excessive MUX usage, increasing complexity and resource consumption. For example, with the HDL code in Listing 2, a good assignment (e.g., assigning (S_2 to S_0) results in 3 MUXs, whereas a poor assignment (S_0 to S_2) results in 7 MUXs. So smaRTLy collects all the inputs of control ports and corresponding outputs, representing them as an Algebraic Decision Diagram (ADD). ADD is a generalization of Binary Decision Diagram (BDD) from $\{0, 1\}$ output sets to arbitrary finite output sets. For example, the output set of Figure 7 is $\{p_0, p_1, p_2, p_3\}$ and Figure 7 is the optimal solution in this case. As BDD is a well-known NP-C problem, so is ADD. Considering the trade-off between performance and efficiency, we use a simple heuristic algorithm: for each MUX, smaRTLy selects the signal that minimizes the total types of terminal nodes of the left and right children. Take Listing 2 as an example, if we select S_2 as the control signal of the first MUX, the number of types of terminal nodes is 4 (left: $\{p_1, p_2, p_3\}$, right: $\{p_0\}$); but if we select S_0, the number is 6 (left: $\{p_0, p_1, p_3\}$, right: $\{p_0, p_1, p_2\}$). In experiments, the algorithm can obtain the optimal solution or close to the optimal solution in most cases. This is because the ADD of actual circuits is usually simple and does not require complex algorithms.

Next, smaRTLy will determine whether to rebuild the muxtree. If we roughly rebuild all muxtrees that meet the line 2 conditions of Algorithm 1, the optimization effect is often poor and may even deteriorate the circuit. This is because while smaRTLy can disconnect the eq gates from the muxtree, they may remain in the circuit if connected to other gates. This connectivity can affect overall circuit performance and complexity. Additionally, a poor assignment or particularly challenging cases might necessitate extra MUXs during reconstruction, which can increase resource usage and circuit area and delay.

To make an informed decision on whether to rebuild the muxtree, we consider several critical factors. First, smaRTLy assesses the potential for removing unnecessary gates and only handles the potential ones. Then, smaRTLy evaluates the height of the rebuilt muxtree, as the height determines the level of complexity in the muxtree structure. The width of the MUX is another important consideration, as a MUX gate is divided into several single-bit gates after technology mapping. smaRTLy also examines the number of additional MUXs needed to maintain the functional equivalence. By taking these factors into account, smaRTLy can decide whether the benefits of muxtree restructuring outweigh the costs. If rebuilding is deemed advantageous, smaRTLy proceed with restructuring according to the optimal assignment. Finally, smaRTLy removes any redundant gates that are no longer connected to the muxtree.

This careful process ensures that the circuit remains optimized, reducing unnecessary components and enhancing overall performance.

IV. Experiment

A. Public Benchmarks

SmaRTLy is implemented using C/C++. All the experiments are implemented on Ubuntu 20.04.6 LTS. The CPU is Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz. The version of GCC is 9.4.0.

We select the top 10 largest circuits written in Verilog from open-source datasets, including IWLS-2005 and RISC-V. For fairness of comparison, we focus on the AIG area, specifically the number of AND gates in the optimized circuit. We exclude Flip-Flop gates from consideration to maintain consistency. We replaced the *opt_muxtree* pass in Yosys with smaRTLy and used the built-in command *aigmap* in Yosys to convert netlists into AIG. All the results generated by our program passed equivalence checking.

The results are presented in Table II. For each case, we first show its original AIG area. Then we show the AIG area of the case optimized by Yosys and smaRTLy, and the last column shows the percentage of area reduction of smaRTLy compared to Yosys, with positive numbers representing improvements. In our benchmark, the range of AIG area is from 23709 to 10836722 and the average is 1415259.6. On average, smaRTLy achieved an extra reduction of 8.95%, which signifies a substantial improvement in logic optimization, as detailed in Table III. This reduction is not very large because Yosys is optimized very well on the public dataset, leaving little room for optimization for smaRTLy. The average optimization of Yosys is 55% and in some cases, it can achieve 94%. Therefore, this reduction is enough to demonstrate the effectiveness of smaRTLy in optimizing circuit designs, leading to more efficient and resource-saving implementations.

We also present the results of two optimizations in smaRTLy. On average, SAT-based redundancy elimination (shown as SAT) achieves a reduction of 3.57%, while muxtree rebuilding (shown as Rebuild) achieves 4.39%. However, the effectiveness of these optimizations can vary significantly between cases.

For instance, in the case of *top_cache_axi*, Rebuild reduces the AIG area by 24.91%, whereas SAT achieves only a 0.01% reduction. Conversely, for *wb_conmax*, SAT achieves a 19.05% reduction, while Rebuild only manages 4.65%. These differences arise from the specific characteristics of each instance. Some examples, with complex data path logic, benefit more from SAT optimization. Others, with numerous case statements, are better suited to Rebuild optimization.

TABLE II: Comparison of AIG areas between Yosys and smaRTLy on benchmark circuits. The first four columns of each row are the case name, the original AIG area, the AIG area after Yosys and smaRTLy optimization. The last column is the proportion of area reduced by smaRTLy compared to Yosys.

Case	Original	Yosys	smaRTLy	Ratio
top_cache_axi	10836722	1301437	977118	24.92%
pci_bridge32	61847	47411	44369	6.42%
wb_conmax	336039	123659	89290	27.79%
mem_ctrl	1118764	65785	65437	0.53%
wb_dma	592158	74697	64322	13.89%
tv80	772802	46137	45070	2.31%
usb_funct	76287	40571	39095	3.64%
ethernet	124127	113507	112202	1.15%
riscv	210141	121280	118689	2.14%
ac97_ctrl	23709	23173	21622	6.69%
Average	1415259.6	195765.7	157721.4	8.95%

TABLE III: Reduction in AIG area by individual methods and their combined effect.

Case	SAT	Rebuild	Full
top_cache_axi	0.01%	24.91%	24.92%
pci_bridge32	0.71%	2.01%	6.42%
wb_conmax	19.05%	4.65%	27.79%
mem_ctrl	0.12%	0.47%	0.53%
wb_dma	11.52%	0.80%	13.89%
tv80	0.71%	1.61%	2.31%
usb_funct	1.60%	1.69%	3.64%
ethernet	0.49%	0.48%	1.15%
riscv	0.17%	1.97%	2.14%
ac97_ctrl	1.34%	5.36%	6.69%
Average	3.57%	4.39%	8.95%

In addition, Full optimization reduces more areas than SAT and Rebuild optimization combined in most cases. This is because SAT and Rebuild optimizations work together to reduce more areas. For example, Rebuild optimization can reduce the height of muxtrees and simplify the control port, which will make the sub-graph smaller in SAT optimization. In logical synthesis, combining multiple optimizations typically yields better results.

B. Industrial Benchmarks

We also tested smaRTLy on an industrial benchmark. Due to confidentiality, we will only show the order of magnitude of the benchmark and the improvement of smaRTLy over Yosys. The average AIG area size of this benchmark is in the millions, of which 37.5% of the test points have an area of more than one million AIG nodes. S

maRTLy can remove 47.2% more AIG areas than Yosys. The result far exceeds the performance of smaRTLy on IWLS-2005 and RISC-V. This is because Yosys performs poorly on this industrial benchmark. In some cases, there is almost no optimization effect. Moreover, the selection circuits are more common in the industrial dataset, so the proportion of MUX gates and PMUX gates is higher. The good performance highlights the efficiency and effectiveness of smaRTLy in processing large-scale industrial applications.

V. Conclusions

In this paper, we propose smaRTLy on multiplexers in the logic synthesis of RTL-level circuits. smaRTLy can deal with more complex relationships between the control ports and remove more gates by rebuilding the muxtree with a better structure. They demonstrated good optimization performance on the public benchmark and industrial benchmark, reducing 8.95% and 47.2% more AIG area than Yosys respectively. In addition, the two optimizations work well together to achieve better performance and have the potential to work with more optimizations in the future.

References

[1] Christoph Albrecht. Iwls 2005 benchmarks. In *International Workshop for Logic Synthesis (IWLS)*, volume 9, 2005.

[2] R Iris Bahar, Erica A Frohm, Charles M Gaona, Gary D Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebric decision diagrams and their applications. *Formal methods in system design*, 10:171–206, 1997.

[3] Lukas Kohutka and Peter Pistek. Faster synthesis of combinational logic based on multiplexer trees and binary decision diagrams. In *2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 239–244. IEEE, 2014.

[4] Shiju Lin, Jinwei Liu, Tianji Liu, Martin DF Wong, and Evangeline FY Young. Novelrewrite: Node-level parallel aig rewriting. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 427–432, 2022.

[5] Tianji Liu, Yang Sun, Lei Chen, Xing Li, Mingxuan Yuan, and Evangeline FY Young. A unified parallel framework for lut mapping and logic optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[6] Tianji Liu and Evangeline FY Young. Rethinking aig resynthesis in parallel. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[7] Dominik Macko and Katarína Jelemenská. Managing digital-system power at the system level. In *2013 Africon*, pages 1–5. IEEE, 2013.

[8] Paul Metzgen and Dominic Nancekievill. Multiplexer restructuring for fpga implementation cost reduction. In *Proceedings of the 42nd Annual Design Automation Conference*, pages 421–426, 2005.

[9] P Pištek, M Kolesár, and K Jelemenská. Optimization of multiplexer trees using modified truth table. In *2010 International Conference on Applied Electronics*, pages 1–4. IEEE, 2010.

[10] Miroslav Siebert and Elena Gramatová. Delay fault coverage increasing in digital circuits. In *2013 Euromicro*

Conference on Digital System Design, pages 475–478. IEEE, 2013.

[11] Niklas Sorensson and Niklas Een. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005(53):1–2, 2005.

[12] Yang Sun, Tianji Liu, Martin DF Wong, and Evangeline FY Young. Massively parallel aig resubstitution. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

[13] Zicheng Wang, Hailong You, Jie Wang, Meihua Liu, Yu Su, and Yong Zhang. Optimization of multiplexer combination in rtl logic synthesis. In *2023 International Symposium of Electronics Design Automation (ISEDA)*, pages 121–125. IEEE, 2023.

[14] Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic, Volume I User level Isa, Andrew Waterman, Yunsup Lee, and David Patterson. The risc-v instruction set manual. *Volume I: User-Level ISA', version*, 2:1–79, 2014.

[15] Clifford Wolf. Yosys open synthesis suite. 2016.

Efficient and Scalable Architectures for Multi-level Superconducting Qubit Readout

Chaithanya Naik Mude[1], Satvik Maurya[1], Benjamin Lienhard[2,3], Swamit Tannu[1]
[1]Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI 53706 USA
[2]Department of Chemistry, Princeton University, Princeton, NJ 08544 USA
[3]Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ 08544 USA

***Abstract*—Realizing the full potential of quantum computing requires large-scale quantum computers capable of running quantum error correction (QEC) to mitigate hardware errors and maintain quantum data coherence. While quantum computers operate within a two-level computational subspace, many processor modalities are inherently multi-level systems. This leads to occasional leakage into energy levels outside the computational subspace, complicating error detection and undermining QEC protocols. The problem is particularly severe in engineered qubit devices like superconducting transmons, a leading technology for fault-tolerant quantum computing. Addressing this challenge requires effective multi-level quantum system readout to identify and mitigate leakage errors. We propose a scalable, high-fidelity three-level readout that reduces FPGA resource usage by $60\times$ compared to the baseline while reducing readout time by 20%, enabling faster leakage detection. By employing matched filters to detect relaxation and excitation error patterns and integrating a modular lightweight neural network to correct crosstalk errors, the protocol significantly reduces hardware complexity, achieving a $100\times$ reduction in neural network size. Our design supports efficient, real-time implementation on off-the-shelf FPGAs, delivering a 6.6% relative improvement in readout accuracy over the baseline. This innovation enables faster leakage mitigation, enhances QEC reliability, and accelerates the path toward fault-tolerant quantum computing.**

I. Introduction

Quantum computing offers the potential for significant computational speedups in fields like quantum chemistry, simulation, cryptography, and optimization, promising advantages over classical systems for tackling complex tasks. However, realizing these speedups depends on the efficient and scalable execution of quantum programs on robust hardware designed to support quantum operations. Quantum information is stored in inherently fragile qubits, the fundamental core units of quantum computation. These are highly susceptible to errors during gate operations due to device imperfections and environmental interference. This vulnerability to errors in qubit operations remains a fundamental challenge to advancing practical quantum technology.

Quantum Error Correction (QEC) can bridge the gap between error-prone qubit devices and practical quantum applications by encoding quantum information as logical qubits across multiple physical qubits to lower the overall error rate when the physical error rate is below a threshold. The effectiveness of QEC grows with redundancy, as measured by the code's distance (d), which exponentially suppresses errors, enabling QEC to achieve the low logical error rates required for practical quantum applications.

Superconducting qubit architectures are among the leading platforms for implementing QEC codes, such as scalable surface codes. QEC uses data qubits to store quantum information and parity qubits for measurements and parity checks, relying heavily on entangling gates and parity qubit measurements to detect and correct errors. A control system manages QEC operations by delivering precise gate pulses, leveraging FPGAs and signal generators for efficient operation in quantum systems with hundreds of qubits.

Readout is a fundamental operation in quantum computing. It is responsible for converting quantum information into classical information within the computational space, represented by the states '0' and '1'. This readout process remains one of the most error-prone and slowest operation, highlighting the ongoing challenges in achieving practical, scalable superconducting quantum processors.

Ideally, qubits in a quantum system should remain within their computational states, labeled '0' and '1.' However, due to the narrow energy gap between these computational states and higher energy levels, as shown in Fig. 1(a), qubits may transition to a higher, non-computational state, known as the leaked state 'L.' These leakage transitions, triggered by thermal excitations, quantum operations, or measurements, push qubits out of the computational basis. Leakage errors disrupt the function of quantum operations, often spreading to neighboring qubits. The effectiveness of QEC relies on precise and timely detection of errors through parity qubit measurements. Slow or inaccurate readout processes increasing the risk of leakage spreading across the system, jeopardizing QEC, potentially blocking the route to quantum advancement. Therefore, fast and effective detection and correction of leakage errors are essential for the reliability of QEC.

To mitigate leakage errors, specialized hardware elements called Leakage Reduction Circuits (LRCs) [3]–[9] are employed to restore qubits to the computational basis, thereby preserving qubit fidelity and maintaining QEC integrity. However, the effectiveness of LRCs hinges on the accuracy of leakage detection; if undetected, leakage can persist and lead to malfunctioning entangling gates employed in QEC circuits. Most LRCs depend on multi-level readout to reliably detect leakage states and apply corrective gates, while others require

C.M., S.M, S.T are supported by The OVCR and WARF's Research Forward initiative and NSF QuSeC-TAQS #2326784, and NSF SaTC #2332405 grants. B.L. is supported by the Swiss National Science Foundation (Postdoc.Mobility Fellowship grant #P500PT_211060).

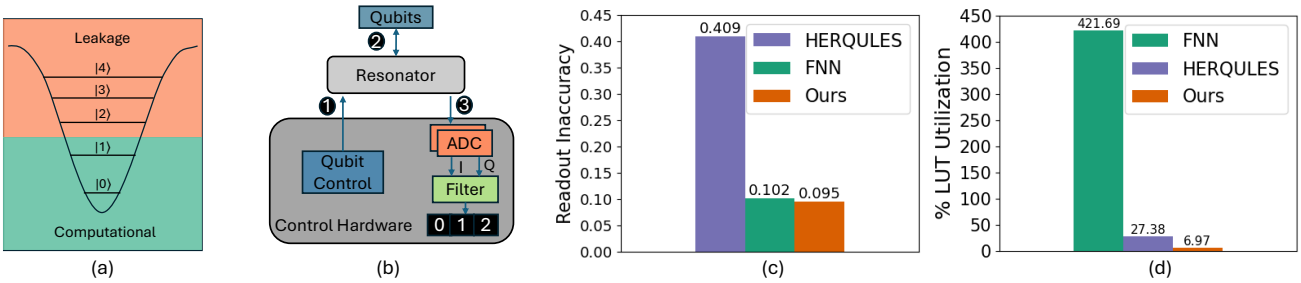


Fig. 1. (a) Computational and Leakage levels in a qubit. (b) Overview of readout pipeline. (c) Comparison of readout classification inaccuracy over all five qubits used in Ref. [1]. (d) LUT utilization of using HERQULES [2], Feedforward Neural Network (FNN) design [1], and our proposed method.

additional specialized control hardware [5]. Multi-level readout also improves speculation of leakage on data qubits [10] using the syndrome measurements of the surface code.

Beyond leakage mitigation, multi-level readout plays a critical role in expanding the capabilities of quantum systems. It enables qudit-based algorithms like efficient Toffoli decompositions [11], [12] and other complex computations. Despite improvements in qubit readout, reset [13], [14], reuse [15], and leakage errors continue to impact performance, highlighting the need for fast, scalable, and reliable multi-level readout to support error correction and advanced qudit algorithms.

Recent advancements in qubit-state readout accuracy have largely been driven by sophisticated discriminators, including deep neural networks (NN) [1], [16], hybrid approaches combining NNs with traditional methods [2] such as matched filters [17], and Hidden Markov Models [18]. Feedforward NNs [1] and autoencoders [19] can directly analyze digitized readout signals without pre-processing, capturing subtle data features that traditional methods often overlook. While these designs enhance state discrimination accuracy, their high computational demands limit scalability for multi-level readout.

While effective for two-level systems, existing solutions FNN [1], HERQULES [2] often struggle to scale with multi-level readout due to large model architectures that face fidelity and hardware efficiency limitations, impacting leakage mitigation. LRCs are essential for addressing leakage errors, but imprecise applications can propagate faults. Fast, reliable leakage mitigation is crucial for fault-tolerant QEC. However, existing designs face two main limitations: large models that are too slow and scalable models [2] that quickly degrade in performance for multi-level systems. The Fig. 1(c), shows that HERQULES is incapable of three-level readout. Additionally, Fig. 1(d) demonstrates that large models require significant FPGA resources, making implementation challenging. Our method uses fewer resources, performing better than the larger FNN model in readout discrimination accuracy.

This manuscript introduces a fast, scalable, and hardware-efficient three-level readout protocol. We reduce model size by almost $100\times$ over FNN [1] and $10\times$ over HERQULES [2], providing quicker inference, enabling scalable, high-fidelity single-shot readout to advance the capabilities of multi-level qubit systems for rapid detection of leakage errors and improve reliability of QEC.

The key contributions of this paper are summarized below:

- We propose a scalable multi-level readout protocol that uses a model size $100\times$ smaller and provides a 6.6% relative improvement in accuracy over the baseline using matched filters and a modular lightweight neural network.
- Our design reduces hardware requirements significantly, utilizing $60\times$ fewer FPGA resources (LookUp Table (LUT)), thereby enabling efficient implementation on the off-the-shelf FPGA hardware.
- We enable a 20% reduction in readout duration, enabling faster and more accurate leakage mitigation to improve overall system reliability.

II. Background

A. Multi-Level Readout for Superconducting Qubits

Multi-level readout is the process of determining a qubit's state post-measurement, typically identifying it as the ground state ('0'), excited state ('1'), or leaked states ('L'). In superconducting qubits, this readout process is enabled by a dispersive coupling between qubits and resonators specifically used for qubit measurement [20], [21].

The readout pipeline, as shown in Fig. 1(b), consists of multiple stages: (1) the control hardware initiates a microwave probe tone sent to the resonator, (2) the qubit's state induces a resonator phase shift picked up by the resonator probe tone, and (3) classical signal processing analyzes the transmitted or reflected readout resonator signal post frequency down-modulation and digitization to infer on the qubit state and assign a ground, excited, or leaked state label. This process is often slow and prone to errors, making precise state inference challenging. Achieving high superconducting-qubit-readout accuracies requires multiple analog components alongside robust signal processing. Here, we focus on enhancing the accuracy and scalability of qubit state discrimination.

ADC. The incoming microwave signal is quadrature modulated, with its In-phase (I) and Quadrature (Q) components retrieved via analog mixing and digitized by two high-speed Analog-to-Digital Converters (ADCs) with typical sampling rates of 250-1000 MSamples/sec.

Filtering. Due to the long measurement times, processing all time-bin samples generated by ADCs for classification is computationally and memory-intensive. Thus, most readout pipelines use a filtering scheme to condense this data. An averaging or a matched filter is commonly applied to reduce the I and Q data streams into a single representative value [17].

Demultiplexing. With frequency-multiplexed readout, qubits are divided into fixed groups to perform readout using the same physical channel. After filtering, the ADC samples are demultiplexed to determine the qubit's state within the group.
Classification. The filtered, demultiplexed samples are used to classify the qubit's state using an appropriate classifier.

III. Impact of Qubit Leakage

A. Gate Malfunction due to Leakage

Recent work by Google Quantum AI [5], [22] shows that leakage errors are among the most significant error sources corrupting the logical qubit information. The impact is comparable to that of errors during Controlled-Z (CZ) gates. The error budget for leakage in CZ gates is similar to that of measurement and reset errors. Studies on superconducting architectures estimate the leakage probability to range from 10^{-4} to 10^{-3}, suggesting that qubit leakage is relatively infrequent and random, making systematic investigations challenging.

We evaluate the effects of leakage experimentally using IBM quantum computers. We employ leakage injection techniques to assess leakage effects on the qubit gate performance, especially in Controlled-NOT (CNOT) gates widely used for surface code syndrome generation. Using the circuit of repeated CNOTs on the IBM Lagos, we initialize the control qubit in the leaked state ($|2\rangle$) and perform 10,000 shots with repeated CNOT operations to measure leakage instances in the target qubit. Results show significantly higher leakage growth of almost $3\times$ within 12 CNOTs with the leaked control qubit necessitating leakage removal.

In a single CNOT gate experiment with a leaked control qubit, with both $|0\rangle$ and $|1\rangle$ as target qubit states, we observed random bit flips and a leakage transfer of 1.5–2% from the control qubit to target qubit after measuring the target qubit.

> The presence of leakage *malfunctions CNOT gates*, necessitating robust leakage mitigation strategies.

B. Impact on Leakage Speculation

The characteristic bit-flip and leakage transport response of leaked qubits on CNOT gates can aid in speculating qubit leakage. Recent work, ERASER [10], utilizes surface code syndrome patterns to speculatively detect leakage and selectively apply LRCs, effectively reducing overall system leakage. Minimizing LRC usage is critical, as unnecessary applications can introduce additional leakage and non-leakage errors due to imperfect LRCs.

TABLE I
Impact of readout on leakage speculation

Design	Accuracy	Leakage Population
ERASER	0.957	4.19 $\times 10^{-3}$
ERASER+M	0.971	2.97 $\times 10^{-3}$

ERASER uses **Multi-level Readout** (ERASER+M) to capture leakage transport, improving leakage speculation accuracy by 2% and leakage population (LP) by $1.5\times$ after 10 QEC cycles for a distance 7 surface code, as shown in Tab. I.

> Multi-level readout enables leakage mitigation improving the performance of quantum error correction.

IV. Scaling High-Fidelity Multi-Level Readout

This section outlines the challenges in achieving scalable, high-fidelity single-shot readout for multi-level quantum systems and the limitations of existing readout methods.

A. Factors affecting Single-Shot Readout Accuracy

The single-shot readout fidelity captures the accuracy of determining a multi-level quantum state in a single measurement and is crucial for reliable quantum computations. Achieving high fidelity is essential for minimizing errors and enabling efficient quantum algorithms. Relaxation and excitation errors, due to crosstalk and unnecessary interactions, limit the fidelity.

Relaxation Errors. Relaxation errors occur due to the spontaneous decay of higher-energy states during readout, caused by qubit-environment interactions. These errors are particularly problematic in long-latency readout operations.

Excitation Errors. Excitation errors can occur when qubits are unintentionally excited to higher energy states during readout. The qubit in the ground state, $|0\rangle$, can get excited to $|1\rangle$ or higher. Similarly $|1\rangle$ can get excited to $|2\rangle$ or higher.

Crosstalk Errors. Readout crosstalk can occur in systems with multiple qubits and readout resonators in close spatial vicinity or frequency spacing. This effect causes the state of neighboring qubits to interfere with readout accuracy. Implementing a robust deep neural network demonstrated substantial error reduction by effectively mitigating the impact of crosstalk [1], [19].

B. Baseline Designs

FNN Design [1]. The intermediate-frequency readout signal is digitized and buffered before reaching a software classifier. Each readout trace, comprising 500 elements per I and Q channel sampled every $2ns$ (totaling $1\mu s$), serves as input to the FNN model [1]. To avoid undersampling, the model uses all ADC samples without demodulation, resulting in an input layer of 1000 neurons. The FNN has 32 outputs, representing the 2^5 basis states of a five-qubit system. For our analysis with a 3-level quantum system, we modify the last layer to 243 outputs, representing 3^5 basis states, as illustrated in Fig. 2.

HERQULES Design [2]. The ADC time-bin samples are demodulated to capture the patterns in traces corresponding to relaxation errors. After filtering and de-multiplexing, individual IQ values and assigned labels are used to obtain matched filters for relaxation and qubit states. This reduces the input size to $2\times$, the number of qubits with the output size as 2^5. With a 3-level quantum system, the input layer increases to $6\times$ number of qubits and the output layer to 243 outputs to represent 3^5 basis states, as illustrated in Fig. 2 (bottom).

Performance Analysis. We examine the performance of state-of-the-art designs mentioned above for three-level quantum system readout. While the HERQULES design outperforms

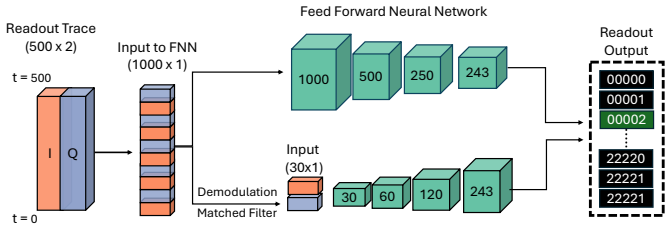


Fig. 2. Design overview of FNN [1](top) and HERQULES [2](bottom)

FNN for two-level readout, it struggles with the increased complexity of three-level readout. In contrast, FNN achieves higher fidelity for three-level readout with a 686 thousand parameter model, but it cannot be efficiently implemented on an FPGA. Tab. II compares the readout fidelity of both designs, revealing performance degradation of HERQULES in handling exponential increase in output states.

TABLE II
THE THREE-LEVEL READOUT FIDELITY OF EXISTING STATE-OF-THE-ART SOLUTIONS WITH $F_{5Q} = \sqrt[5]{F_1F_2F_3F_4F_5}$

Design	Qubit 1	Qubit 2	Qubit 3	Qubit 4	Qubit 5	F_{5Q}
FNN	0.967	0.728	0.927	0.932	0.962	0.898
HERQULES	0.598	0.549	0.608	0.607	0.594	0.591

C. Challenges with Existing Methods

Readout Accuracy. For multi-level systems, achieving high readout accuracy is challenging due to the complexity of distinguishing the exponentially large number of states compared to two-level systems. HERQULES struggles with three-level systems due to a limited model capacity, and the FNN is impractical for real-time use due to its large parameter count and high hardware-demands highlighting the need for an accurate, hardware-efficient multi-level readout discriminator.

Hardware Complexity. The computational demands for implementing the FNN and HERQULES designs grow with system size. For a system of n qubits with k-levels each, the output layer scales exponentially as k^n increasing total neural network parameters of the model. Additionally for HERQULES, the input layer scales as $O(nk^2)$, growing quadratically with k and linearly with n due to the relaxation and error matched filters required between each pair of k-levels for each qubit.

Readout Latency. As quantum systems scale in the number of qubits or qudits, managing readout latency becomes critical for maintaining computational efficiency. Longer latencies can impair performance, particularly in large systems where timely feedback is essential for effective error correction and system stability. With increasing neural network parameters, inference latency also rises, limiting model designs for larger qubit counts. Since the output layer scales as k^n, models that scale linearly with the number of qubits (n) are essential for practical implementation.

Qubit Leakage Calibration. Detection of leakage traces by calibrating them in a leaked state adds additional gate engineering steps and increases the resource overhead further.

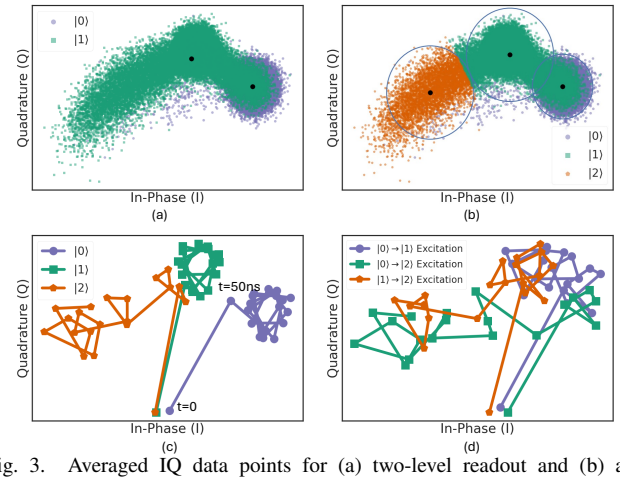


Fig. 3. Averaged IQ data points for (a) two-level readout and (b) after detecting instances of natural leakage using spectral clustering. Mean traces of the clusters of (c) qubit states and (d) excitation error instances

V. Enabling Efficient Multi-Level Readout

In this paper, we propose an architecture that tackles the challenges posed in the previous section and focuses on improving fidelity, reducing latency, and enhancing the scalability of superconducting quantum system readout.

A. Detecting Leakage Cluster without Explicit Calibration

Calibrating qubits to populate leaked states is a complex process. Although rare, naturally occurring leakage in standard two-level readout traces reflects the probabilistic distribution of all leakage states more accurately than explicit calibration. These traces more accurately reflect the behavior of leaked qubits and can be distinguished from readout traces, as the averaged characteristics of readout traces for each state typically form distinct clusters.

We calculate the Mean Trace Value (MTV) to identify distinct clusters of qubit states as mentioned in HERQULES [2]. For a trace Tr, MTV is defined as $MTV = \frac{1}{len(Tr)} \sum_{t=0}^{len(Tr)} Tr(t)$. This temporal mean of each trace corresponds to a single point in Fig. 3(a). Differences in the mean trace patterns across states suggest that readout trace-level information can enhance qubit state discrimination by leveraging inherent data patterns.

MTV points can be used to identify leaked states through spectral clustering into three classes. Most traces will correspond to computational states, while the smallest cluster will likely represent leaked states. Spectral clustering outputs three unlabeled clusters, which can then be labeled based on the probability of leakage when the state is prepared in a computational state, enabling accurate label assignment. As shown in Fig. 3(b), this approach identifies naturally occurring leakage states without needing explicit calibration.

B. Matched Filter for Multi-Level Classification

In single-qubit-state readout, matched filtering is a standard tool tomaximize the signal-to-noise ratio (SNR) [17]. Using statistical properties of signal traces, we define the Matched Filter (MF) kernel K as the mean difference of traces normalized by variance differences, enhancing state discrimination by

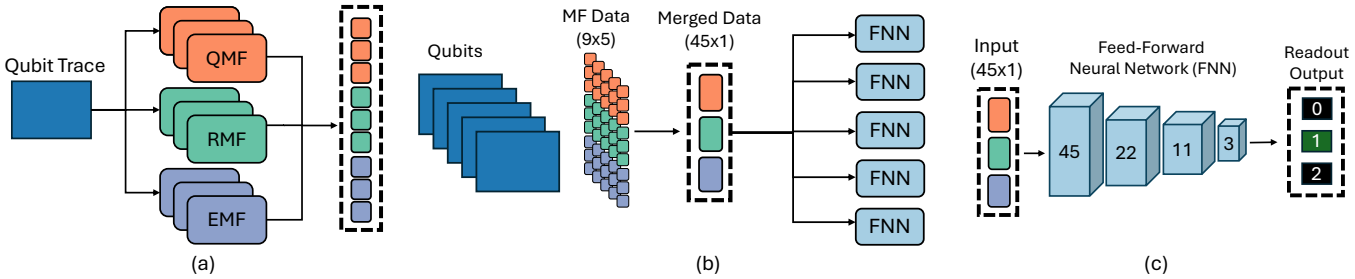


Fig. 4. Overview of our Design. (a) The qubit traces are processed with matched filter envelopes by dot-product to output a scalar value for each envelope. (b) The data from all the qubits are merged to send the data as the input to a Feed-Forward Neural Network (c) Our design of FNN to output qudit states.

inversely weighting trace differences by variance. Let μ_0 and μ_1 represent the mean of traces corresponding to two distinct quantum states, with σ_0^2 and σ_1^2 as their respective variances. Then kernel is as $K = \frac{\mu_1 - \mu_0}{\sigma_1^2 - \sigma_0^2}$.

Most MFs are optimized for binary classification, relying on the statistical distinction between two states to maximize SNR. Higher-order MFs are mathematically very cumbersome. To address this, we use three two-state MF tailored to specific classes, but residual errors persist due to their limitations in separating multi-level states. To improve accuracy, we employ a small neural network to handle non-linearities.

Deciphering Error Traces and Error Matched Filters. The centroids of each state serve as priors for cluster identification. Traces belonging to a particular state but positioned closer to other cluster centroids can be tagged as error traces. With estimated centroids and state traces, we can use this information and ground truth data to label traces corresponding to relaxation and excitation events. Fig. 3(d) shows the MTV for such excitation traces from $|0\rangle$ to $|1\rangle$ and $|2\rangle$ and from $|1\rangle$ to $|2\rangle$. Quantum state discrimination can be improved by learning characteristic patterns exhibited by these error traces.

C. Our Design

The architecture of the FNN baseline design and HERQULES learns features corresponding to crosstalk, qubit decay, and other non-idealities to achieve high-fidelity qubit readout. Unfortunately, they require significant computational and memory resources. We enable a hardware-efficient design that scales favourably with the increasing number of qubits.

The broad overview of the design is discussed in Fig. 4. Demodulated[1] ADC data is used to train MFs, including Qubit MF (QMF), Relaxation MF (RMF), and Excitation MF (EMF), as discussed in Tab. III. We incorporate a small NN to handle remaining non-linearities, yielding our design. The NN structure, shown in Fig. 4(c), has an input size (P) that scales as $O(nk^2)$ for n qubits with k-levels, as each qubit requires $O(k^2)$ error MFs and the NN has two hidden layers with size $\lfloor P/2 \rfloor$, $\lfloor P/4 \rfloor$ and output size of k. For three-level systems, we have three QMFs, RMFs, and EMFs for each pair of levels summing up to an input size of 45 for our design.

In contrast to HERQULES, which classifies all qubits collectively, our approach processes each qubit's output individually while incorporating information from all qubits, resulting in k outputs per qubit rather than k^n. This method enables polynomial growth in (n, k) for model size rather than exponential, making it resource-friendly for larger qubit systems.

[1]demodulation is fast and in-expensive requiring two FMA units

TABLE III
OVERVIEW OF EMPLOYED MATCHED FILTERS

Matched Filter	To Distinguish
Qubit Matched Filter (QMF)	$\|0\rangle, \|1\rangle, \|2\rangle$
Relaxation Matched Filter (RMF)	$\|1\rangle \rightarrow \|0\rangle, \|2\rangle \rightarrow \|0\rangle, \|2\rangle \rightarrow \|1\rangle$
Excitation Matched Filter (EMF)	$\|0\rangle \rightarrow \|1\rangle, \|0\rangle \rightarrow \|2\rangle, \|1\rangle \rightarrow \|2\rangle$

D. Training Details

MFs are based on the mean and variance of labeled readout traces with the aim to maximize the SNR for each state. We create qubit-level MFs for each pair of qubit states, as well as error MFs: RMFs for relaxation traces and EMFs for excitation traces. During inference, these kernels are applied to incoming readout traces (Fig. 4(a)) to generate likelihood scores for each state, which serve as inputs for further refinement. To address non-linearities, a NN is trained for each qubit on the outputs of qubit MFs, RMFs, and EMFs from all qubits using labeled data to optimize classification boundaries as shown in Fig. 4(b). During inference, the NN processes these outputs as shown in Fig. 4(c), refining state predictions, addressing overlaps, and enhancing multi-level readout accuracy.

VI. METHODOLOGY

Quantum Hardware. We obtained datasets containing the readout time traces collected directly from the ADC originating from a five-qubit chip used in Ref. [1]. These qubits are read out via individual readout resonators coupled to a common feedline using frequency-multiplexing. The ADC sampling rate is 500 MSamples/sec, and qubit relaxation (T_1) times range from $7\mu s$ to $40\mu s$.

The dataset contains readout traces for all 32 basis states of the five qubits, with 50,000 traces per basis state (32 × 50000 = 1600000 traces). We fixed the readout duration to $1\mu s$ for all qubits. Additionally, we use the third and fourth qubits, which are more prone to $|2\rangle$ excitations, to understand the impact of the excitation-matched filter.

After spectral clustering, the total traces for computational and leaked states vary for each qubit, from the lowest of 487 traces for Qubit 1 to 17,642 for Qubit 4. We divide the train and test as 30-70 split for each of the 3^5 possible states and

use 15% of the training dataset as the validation dataset. The distinguishability of the states of qubit 2 is limited due to the experimental setup in Ref. [1].

FPGA Hardware. To estimate the FPGA resources needed to implement a NN, we use a combination of the `hls4ml` [23] tool and Xilinx Vivado High-Level Synthesis (HLS). `hls4ml` can take a NN model written in frameworks such as Keras or Pytorch and create an equivalent HLS model that can then be synthesized with Vivado HLS. We use the Xilinx Zynq MPSoC `xczu7ev-ffvc1156-2-i` as the target device.

VII. Evaluations

A. Impact on Readout Fidelity

Tab. IV presents the readout fidelity for the modified FNN design and our proposed method, showing a relative improvement of 6.6% ($= \frac{90.52-89.85}{100-89.85}$). The FNN requires almost $85\times$ more LUTs than our method.

TABLE IV
THE THREE-LEVEL READOUT FIDELITY OF ALL 3^5 STATES WITH CUMULATIVE ACCURACY $F_{5Q} = \sqrt[5]{F_1F_2F_3F_4F_5}$

Design	QUBIT 1	QUBIT 2	QUBIT 3	QUBIT 4	QUBIT 5	F_{5Q}
FNN	0.967	0.728	0.928	0.932	0.962	0.8985
OURS	0.971	0.745	0.923	0.939	0.969	0.9052

Qubit 3 and 4 are more prone to leakage, we want to compare our methods with existing single qubit methods. Tab. V compares the readout fidelity of the discriminant-analysis based methods (LDA, QDA) and our proposed method. Our design demonstrates a $1-2\%$ improvement over NN and upto 6% over LDA. This improvement is mainly attributed to additional information on relaxation and excitation errors.

TABLE V
THE THREE-LEVEL READOUT FIDELITY OF SINGLE-QUANTUM STATES

Design	**LDA**	**QDA**	**NN**	**OURS**
Qubit 3	0.8966	0.914	0.939	0.959
Qubit 4	0.9181	0.921	0.926	0.930

B. Impact on Readout Latency and QEC Cycle Time

We enable faster readout by reducing the readout time by 200 ns without much loss in the overall discrimination accuracy across all qubits at varying trace lengths, as shown in Fig. 5b). Qubit-state readout, typically the slowest operation in QEC cycles, significantly reduces the QEC performance. The measurement-time reduction yields up to a 17% decrease in QEC cycle time[2] for the surface-17 circuit [24], providing a valuable tradeoff for systems with large code distances and directly decreasing total execution time.

C. FPGA Resource Utilization

The FNN design requires $60\times$ ($\approx \frac{420}{7}$) more LUT utilization on an FPGA than our design and $15\times$ ($\approx \frac{420\%}{28\%}$) more than HERQULES. Our design requires significantly lower FPGA resource utilization than HERQULES, as illustrated in Fig. 5(a), with the key metrics as LUTs, Flip-Flops (FF), Block RAM (BRAM), and Digital Signal Processing (DSP) units, with over $5\times$ reduction in FFs and $4\times$ LUTs compared to HERQULES , indicating scalability of our approach.

[2]QEC requires repeated measurements impacting execution time for quantum algorithms

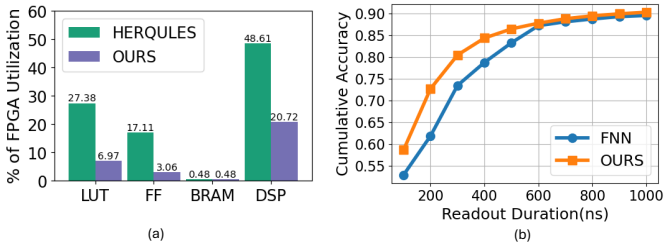


Fig. 5. (a) Comparison of FPGA resource utilization (b)Variation of mean accuracy with readout duration in nanoseconds

D. Power Consumption

The Synopsys design compiler is used to evaluated the power consumption using a 45nm TSMC standard cell library. With our design we require 1.561 mW total power at a 1 GHz clock rate and a latency of 5 cycles (5 ns).

E. Impact on Leakage Speculation

Our method and the FNN has fewer readout errors than discriminant-analysis-based methods, such as QDA and LDA. The accuracy of leakage speculation improves significantly as readout error decreases, rising from 0.913 to 0.947 as shown in Tab. VI. While the FNN outperforms QDA and LDA in speculation accuracy, it requires more inference time. Our method surpasses the FNN in both accuracy and speed, due to a $100\times$ smaller model size, enabling faster leakage detection and improving overall system performance.

ERASER+M is run for 10 QEC cycles for a surface code to obtain speculation accuracy mentioned in Tab. VI. We calculate the error as the infidelity of mean accuracy excluding Qubit 2 due to experimental limitations during its setup.

TABLE VI
IMPACT OF MULTI-LEVEL READOUT ON LEAKAGE SPECULATION

Design	**Error(%)**	**Speed**	**Speculation Accuracy**
LDA	**10**	Fast	0.914
QDA	**9**	Fast	0.921
FNN	**5.5**	Slow	0.943
Ours	**5**	Fast	0.947

VIII. Conclusion

We present a scalable, hardware-efficient qudit-state-readout protocol that combines matched filters with lightweight neural networks to achieve high accuracy and efficient leakage mitigation. By transitioning the scaling of the neural network architecture from exponential to polynomial, our approach reduces hardware demands and enables practical FPGA deployment. Additionally, enabling fast readout with a 20% reduction in readout duration accelerates performance without requiring additional training. This multi-level readout design strengthens QEC by enabling effective speculation for fast leakage detection, advancing reliable, fault-tolerant quantum systems and moving closer to efficient quantum processors.

REFERENCES

[1] B. Lienhard, A. Vepsäläinen, L. C. Govia, C. R. Hoffer, J. Y. Qiu, D. Ristè, M. Ware, D. Kim, R. Winik, A. Melville, B. Niedzielski, J. Yoder, G. J. Ribeill, T. A. Ohki, H. K. Krovi, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "Deep-neural-network discrimination of multiplexed superconducting-qubit states," *Phys. Rev. Appl.*, vol. 17, p. 014024, Jan 2022.

[2] S. Maurya, C. N. Mude, W. D. Oliver, B. Lienhard, and S. Tannu, "Scaling qubit readout with hardware efficient machine learning architectures," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, ACM, June 2023.

[3] M. Suchara, A. W. Cross, and J. M. Gambetta, "Leakage suppression in the toric code," in *2015 IEEE International Symposium on Information Theory (ISIT)*, pp. 1119–1123, 2015.

[4] J. Ghosh, A. G. Fowler, J. M. Martinis, and M. R. Geller, "Understanding the effects of leakage in superconducting quantum-error-detection circuits," *Physical Review A*, vol. 88, Dec. 2013.

[5] K. C. Miao, M. McEwen, J. Atalaya, D. Kafri, L. P. Pryadko, A. Bengtsson, A. Opremcak, K. J. Satzinger, Z. Chen, P. V. Klimov, C. Quintana, R. Acharya, K. Anderson, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. C. Bardin, A. Bourassa, J. Bovaird, L. Brill, B. B. Buckley, D. A. Buell, T. Burger, B. Burkett, N. Bushnell, J. Campero, B. Chiaro, R. Collins, P. Conner, A. L. Crook, B. Curtin, D. M. Debroy, S. Demura, A. Dunsworth, C. Erickson, R. Fatemi, V. S. Ferreira, L. F. Burgos, E. Forati, A. G. Fowler, B. Foxen, G. Garcia, W. Giang, C. Gidney, M. Giustina, R. Gosula, A. G. Dau, J. A. Gross, M. C. Hamilton, S. D. Harrington, P. Heu, J. Hilton, M. R. Hoffmann, S. Hong, T. Huang, A. Huff, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, J. Kelly, S. Kim, F. Kostritsa, J. M. Kreikebaum, D. Landhuis, P. Laptev, L. Laws, K. Lee, B. J. Lester, A. T. Lill, W. Liu, A. Locharla, E. Lucero, S. Martin, A. Megrant, X. Mi, S. Montazeri, A. Morvan, O. Naaman, M. Neeley, C. Neill, A. Nersisyan, M. Newman, J. H. Ng, A. Nguyen, M. Nguyen, R. Potter, C. Rocque, P. Roushan, K. Sankaragomathi, H. F. Schurkus, C. Schuster, M. J. Shearn, A. Shorter, N. Shutty, V. Shvarts, J. Skruzny, W. C. Smith, G. Sterling, M. Szalay, D. Thor, A. Torres, T. White, B. W. K. Woo, Z. J. Yao, P. Yeh, J. Yoo, G. Young, A. Zalcman, N. Zhu, N. Zobrist, H. Neven, V. Smelyanskiy, A. Petukhov, A. N. Korotkov, D. Sank, and Y. Chen, "Overcoming leakage in quantum error correction," *Nature Physics*, vol. 19, p. 1780–1786, Oct. 2023.

[6] B. G. Markaida and L.-A. Wu, "Implementation of leakage elimination operators and subspace protection," *Scientific Reports*, vol. 10, p. 18846, Nov 2020.

[7] F. Battistel, B. Varbanov, and B. Terhal, "Hardware-efficient leakage-reduction scheme for quantum error correction with superconducting transmon qubits," *PRX Quantum*, vol. 2, July 2021.

[8] C. J. Wood and J. M. Gambetta, "Quantification and characterization of leakage errors," *Physical Review A*, vol. 97, Mar. 2018.

[9] M. McEwen, D. Kafri, Z. Chen, J. Atalaya, K. J. Satzinger, C. Quintana, P. V. Klimov, D. Sank, C. Gidney, A. G. Fowler, F. Arute, K. Arya, B. Buckley, B. Burkett, N. Bushnell, B. Chiaro, R. Collins, S. Demura, A. Dunsworth, C. Erickson, B. Foxen, M. Giustina, T. Huang, S. Hong, E. Jeffrey, S. Kim, K. Kechedzhi, F. Kostritsa, P. Laptev, A. Megrant, X. Mi, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Niu, A. Paler, N. Redd, P. Roushan, T. C. White, J. Yao, P. Yeh, A. Zalcman, Y. Chen, V. N. Smelyanskiy, J. M. Martinis, H. Neven, J. Kelly, A. N. Korotkov, A. G. Petukhov, and R. Barends, "Removing leakage-induced correlated errors in superconducting quantum error correction," *Nature Communications*, vol. 12, Mar. 2021.

[10] S. Vittal, P. Das, and M. Qureshi, "Eraser: Towards adaptive leakage suppression for fault-tolerant quantum computing," in *56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, ACM, Oct. 2023.

[11] A. Litteken, L. M. Seifert, J. D. Chadwick, N. Nottingham, T. Roy, Z. Li, D. Schuster, F. T. Chong, and J. M. Baker, "Dancing the quantum waltz: Compiling three-qubit gates on four level architectures," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, ACM, June 2023.

[12] A. S. Nikolaeva, E. O. Kiktenko, and A. K. Fedorov, "Generalized toffoli gate decomposition using ququints: Towards realizing grover's algorithm with qudits," *Entropy*, vol. 25, p. 387, Feb. 2023.

[13] M. DeCross, E. Chertkov, M. Kohagen, and M. Foss-Feig, "Qubit-Reuse Compilation with Mid-Circuit Measurement and Reset," *Physical Review X*, vol. 13, p. 041057, Oct. 2023.

[14] K. J. Mesman, F. Battistel, E. Reehuis, D. de Jong, M. J. Tiggelman, J. Gloudemans, J. C. van Oven, and C. C. Bultink, "Q-profile: Profiling tool for quantum control stacks applied to the quantum approximate optimization algorithm," 2023.

[15] S. Brandhofer, I. Polian, and K. Krsulich, "Optimal qubit reuse for near-term quantum computers," 2023.

[16] U. Azad and H. Zhang, "Machine learning based discrimination for excited state promoted readout," 2022.

[17] C. A. Ryan, B. R. Johnson, J. M. Gambetta, J. M. Chow, M. P. da Silva, O. E. Dial, and T. A. Ohki, "Tomography via correlation of noisy measurement records," *Phys. Rev. A*, vol. 91, p. 022118, Feb 2015.

[18] B. M. Varbanov, F. Battistel, B. M. Tarasinski, V. P. Ostroukh, T. E. O'Brien, L. DiCarlo, and B. M. Terhal, "Leakage detection for a transmon-based surface code," *npj Quantum Information*, vol. 6, Dec. 2020.

[19] P. Luchi, P. E. Trevisanutto, A. Roggero, J. L. DuBois, Y. J. Rosen, F. Turro, V. Amitrano, and F. Pederiva, "Enhancing qubit readout with autoencoders," *Phys. Rev. Appl.*, vol. 20, p. 014045, Jul 2023.

[20] A. Blais, R.-S. Huang, A. Wallraff, S. M. Girvin, and R. J. Schoelkopf, "Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation," *Phys. Rev. A*, vol. 69, p. 062320, Jun 2004.

[21] R. Yanagimoto, R. Nehra, R. Hamerly, E. Ng, A. Marandi, and H. Mabuchi, "Quantum nondemolition measurements with optical parametric amplifiers for ultrafast universal quantum information processing," *PRX Quantum*, vol. 4, Mar. 2023.

[22] R. Acharya, I. Aleiner, R. Allen, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, R. Babbush, D. Bacon, J. C. Bardin, J. Basso, A. Bengtsson, S. Boixo, G. Bortoli, A. Bourassa, J. Bovaird, L. Brill, M. Broughton, B. B. Buckley, D. A. Buell, T. Burger, B. Burkett, N. Bushnell, Y. Chen, Z. Chen, B. Chiaro, J. Cogan, R. Collins, P. Conner, W. Courtney, A. L. Crook, B. Curtin, D. M. Debroy, A. Del Toro Barba, S. Demura, A. Dunsworth, D. Eppens, C. Erickson, L. Faoro, E. Farhi, R. Fatemi, L. Flores Burgos, E. Forati, A. G. Fowler, B. Foxen, W. Giang, C. Gidney, D. Gilboa, M. Giustina, A. Grajales Dau, J. A. Gross, S. Habegger, M. C. Hamilton, M. P. Harrigan, S. D. Harrington, O. Higgott, J. Hilton, M. Hoffmann, S. Hong, T. Huang, A. Huff, W. J. Huggins, L. B. Ioffe, S. V. Isakov, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, P. Juhas, D. Kafri, K. Kechedzhi, J. Kelly, T. Khattar, M. Khezri, M. Kieferová, S. Kim, A. Kitaev, P. V. Klimov, A. R. Klots, A. N. Korotkov, F. Kostritsa, J. M. Kreikebaum, D. Landhuis, P. Laptev, K.-M. Lau, L. Laws, J. Lee, K. Lee, B. J. Lester, A. Lill, W. Liu, A. Locharla, E. Lucero, F. D. Malone, J. Marshall, O. Martin, J. R. McClean, T. McCourt, M. McEwen, A. Megrant, B. Meurer Costa, X. Mi, K. C. Miao, M. Mohseni, S. Montazeri, A. Morvan, E. Mount, W. Mruczkiewicz, O. Naaman, M. Neeley, C. Neill, A. Nersisyan, H. Neven, M. Newman, J. H. Ng, A. Nguyen, M. Nguyen, M. Y. Niu, T. E. O'Brien, A. Opremcak, J. Platt, A. Petukhov, R. Potter, L. P. Pryadko, C. Quintana, P. Roushan, N. C. Rubin, N. Saei, D. Sank, K. Sankaragomathi, K. J. Satzinger, H. F. Schurkus, C. Schuster, M. J. Shearn, A. Shorter, V. Shvarts, J. Skruzny, V. Smelyanskiy, W. C. Smith, G. Sterling, D. Strain, M. Szalay, A. Torres, G. Vidal, B. Villalonga, C. Vollgraff Heidweiller, T. White, C. Xing, Z. J. Yao, P. Yeh, J. Yoo, G. Young, A. Zalcman, Y. Zhang, N. Zhu, and G. Q. AI, "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, pp. 676–681, Feb 2023.

[23] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. P. Carloni, G. D. Guglielmo, P. Harris, J. Krupa, D. Rankin, M. B. Valentin, J. Hester, Y. Luo, J. Mamish, S. Orgrenci-Memik, T. Aarrestad, H. Javed, V. Loncar, M. Pierini, A. A. Pol, S. Summers, J. Duarte, S. Hauck, S.-C. Hsu, J. Ngadiuba, M. Liu, D. Hoang, E. Kreinar, and Z. Wu, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," 2021.

[24] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. DiCarlo, "Scalable quantum circuit and control for a superconducting surface code," *Physical Review Applied*, vol. 8, Sept. 2017.

WISEDRAM: A Reliable Bitwise In-DRAM Accelerator

Mohammad Arman Soleimani[1], Nezam Rohbani[2,3], Adrian Cristal Kestelman[2], Osman Unsal[2], Hamid Sarbazi-Azad[1,3]
[1]Department of Computer Engineering, Sharif University of Technology, Iran
[2]Barcelona Supercomputing Center, Spain
[3]Institute for Research in Fundamental Sciences (IPM), Iran
{arman.soleimani, azad}@sharif.edu {nrohbani, osman.unsal, adrian.cristal}@bsc.es

***Abstract*—Processing-in-Memory (PIM) aims to address the costly data movement between processing elements and memory subsystem, by computing simple operations inside DRAM in parallel. The large capacity, wide activation size during cell access, and the maturity of DRAM technology, make this technology a great choice for PIM techniques. Nonetheless, vulnerability to process variation and noises, internal leakage of the cells, and high latency in cell access, limit the utilization of processing in DRAMs for real-world applications. This work proposes a fast PIM technique, called WISEDRAM, which leverages one row of special cells, called X-cells, to enable in-DRAM bulk-bitwise operations. Unlike previous approaches, WISEDRAM retains the conventional DRAM cell access procedure, thereby ensuring the reliability of cell access for reads and writes at a level equivalent to that of conventional DRAMs. Compared with the state-of-the-art, WISEDRAM exhibits 22% reduction in average bitwise computation latency and a 71% improvement in XOR operation execution speed, while imposing an area overhead of 1.6%.**

***Index Terms*—DRAM, Processing-In-Memory, Bitwise Operation, Memory Wall**

I. Introduction

Transferring data between memory and processing elements, such as the CPU or GPU is energy-intensive and suffers from high latency [12], [22]. Furthermore, due to limited datapath width, i.e. 64 bit in DDR5, memory access is serialized, which exacerbates memory access latency. This problem, which results in a performance bottleneck, is so severe that it has been termed the "memory wall". Memory-intensive applications, such as neural network processing and graph processing, are particularly vulnerable to the memory wall effect, and unpredictable memory accesses further worsen the problem by reducing the effectiveness of cache hierarchy [6], [23].

To reduce the amount of data transmission between the memory subsystem and processing cores, various techniques have been proposed, including Near Memory Processing (NMP) and Processing In Memory (PIM) [21], [24].

In NMP, processing logic, such as small processing units, is placed near the memory. This reduces the latency between the two units, and relaxes the memory wall but does not entirely eliminate it. There is data movement between the processing logic and memory, thus the problem still fundamentally exists. Moreover, NMP schemes cannot fully harness the parallelism that exists in memory arrays and offer less throughput potential than processing directly on the memory arrays [24].

On the other hand, PIM involves enabling processing inside memory arrays and performing operations on stored data inside the memory. This can be achieved by adding logic to each column of memory, or by exploiting analog properties and available memory elements. Unlike NMP, PIM is able to fully exploit the huge amounts of parallelism available inside memory arrays, and completely eliminate the memory wall, as there is no processor-memory data movement.

Various PIM techniques have been presented for different types of memory technologies, including Non-Volatile Memories (NVMs) [26], ReRAMs [4], SRAMs [2], and DRAMs [28]. The 1-transistor, 1-capacitor DRAM (1T1C-DRAM) structure has gained significant attention for PIM due to its high capacity and massive internal parallelism, maturity of its technology, and the costly off-chip data movement between main memory and processor. The memory wall is often observed between DRAM and the processor, since on-chip cache has considerably less latency compared to off-chip DRAM, and the vast majority of computer systems rely on DRAM as main memory.

However, a practical implementation of PIM in conventional DRAM devices faces unique challenges. The highly compact nature of the 1T1C-DRAM arrays, with bitline pitches of approximately 3 F to 4 F (F = Feature size) [14], makes it extremely difficult to accommodate additional PIM-related circuits or modifications, as these must be compatible with the DRAM fabrication process. This poses a significant challenge, as DRAM chips are fabricated with less metal layers compared with typical CMOS chips which implement processing logic [14], [32]. As a result, integrating logic into DRAM chips is often complex and challenging.

This work presents WISEDRAM, a novel technique for in-DRAM processing, which aims to improve upon the state-of-the-art. The proposed method adds one row of a special dual-contact memory cell, called X-cell, to the DRAM array. This row of cells is capable of performing any two-operand bitwise operation in three DRAM cycles, which is the least amount of cycles theoretically possible - two operand reads and one result write - and significantly faster than current methods.

WISEDRAM's fast and robust computations are shown to greatly improve upon the state-of-the-art in terms of robustness, and our simulations show that this is a strong concern with the state-of-the-art. The approach of enabling computation through a row of specialized cells means that other parts of the memory structure, such as the decoders, peripherals, and amplifiers, are not modified. Thus, WISEDRAM is less intrusive in terms of impact on chip development.

To make the case for WISEDRAM's design and the X-Cell, we conduct a qualitative and quantitative comparison against four state-of-the-art PIM techniques, Ambit [31], ROC [33], ELP2IM [34], and PIPF-DRAM [28], and our technique outperforms these techniques in terms of performance, particularly in performing X(N)OR operation, with a speed improvement of up to 71%. On average, WISEDRAM is faster than the

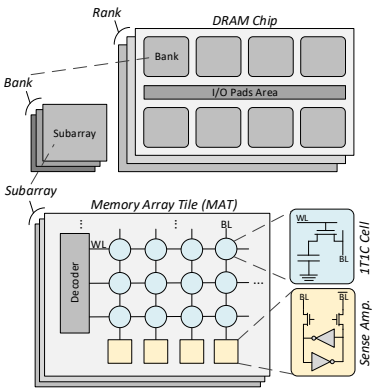


Fig. 1: DRAM chip general structure.

fastest technique presented in the literature to perform bulk-bitwise operations by 22%. WISEDRAM computes any two-operand bitwise operation in three DRAM cycles, which is the least possible number of cycles. Additionally, WISEDRAM is more robust against process variation, achieving a robustness improvement of 77% when compared with the most robust processing in DRAM technique in the literature. The proposed technique incurs a negligible area overhead of only 1.6%.

II. Preliminaries

A. DRAM Fundamentals

1) DRAM Structure: The most widely used DRAM cell structure is 1T-1C (One Transistor-One Capacitor) configuration, where each memory cell consists of a capacitor as the storage element and an access transistor [14].

In the typical structure of DRAM memories, the memory cells are arranged in a two-dimensional layout within a Memory Array Tile (MAT), with dimensions such as 512×512 or 1024×1024 memory cells. Access to these memory cells is facilitated through horizontal and vertical wires known as Wordlines (WLs) and Bitlines (BLs), respectively [5], [19]. Local wordline drivers are activated by a row clock/address spine, which is in turn controlled by global wordlines. A differential *sense amplifier* (SA) at each bitline pair detects the tiny voltage perturbations during cell access. Notably, adjacent MATs in the DRAM chip share SAs as well as wordline drivers.

An overview of DRAM's structure hierarchy can be seen in Figure 1. Each DRAM chip comprises multiple banks, which in turn comprise multiple sub-arrays. Finally, each sub-array is made up of multiple MATs, described earlier.

2) DRAM Operation: Three main operations in DRAM memories - Read, Write, and Refresh - are closely related. In Read and Write, data is transferred in opposite directions, whereas in a Refresh operation - required to restore the leaked charge in DRAM cells - only cell access occurs without data transfer [14]. Prior to accessing a cell, all the bitlines are precharged to half-VDD (*VDD/2*) and then left floating. This phase is known as the *precharge* phase in DRAM cells access. Then, to access a cell during the *activation* phase, the corresponding wordline is activated, causing all the memory cells, each consisting of a tiny capacitor, to connect to the corresponding bitlines. This results in a voltage perturbation (V_S) of about 100 mV on the floated bitline by charge sharing.

Due to the environmental and internal noise, differential sensing is essential to reliably read the V_S value. Thus, bitlines are arranged in pairs connected to differential SAs. One of the bitlines in each pair is left floating, with its voltage expected to be at *VDD/2*, and this voltage is used to detect positive or negative voltage perturbations on the other bitline in the pair. Since the stored charge in the accessed cell is either *0* or *VDD*, the voltage of the bitline connected to the cell will be either lower or higher than the reference (floating) bitline. Using this scheme, the SA cancels out potential noise on the bitline pair and reliably detects the value on the accessed cell.

The structure of a differential SA, as illustrated in Figure 1, typically consists of cross-coupled inverters with strong positive feedback, allowing voltage comparison. The input/output with a higher voltage is rapidly pulled to *VDD*, while the other one is pulled to *GND*. During the precharge phase, the SA power is gated from the VDD and GND power rails, resulting in the internal nodes of the SA being precharged to *VDD/2* alongside the bitlines.

To prevent voltage drop while charging or discharging bitcell capacitors to/from *VDD* using the NMOS access transistor, the controlling voltage of wordlines should be set higher than *VDD*+V_{Th}, where V_{Th} represents the threshold voltage of the NMOS access transistor. This higher voltage level is known as the High-Level VDD or V_{DH}. In the case of DDR5, this voltage is provided by the 1.8 V input pins, also known as *VPP* [1].

When a cell is connected to a bitline, the stored data/charge in the cell is destroyed due to charge sharing operation. To restore the cell charge, the corresponding bitline, which is connected to the accessed cell, is charged/discharged to the same charge level as the cell (before the access) during the stabilization phase of the SA [7], [14].

Once the SA is stabilized to around 3/4 of *VDD*, data can be accessed by activating the Column Select Line [7], [27]. The data is differentially transferred to the global sense amplifier to improve the speed and reliability of data transfer, and the data becomes available at the output.

B. Related Work

PIM techniques that directly utilize DRAM cells for processing can be divided into two categories: Analog Processing In-DRAM and Semi-Digital Processing In-DRAM. Other techniques, such as Logic-Placement In-DRAM [5], [15], [17], [35] and In-DRAM Look-Up Tables, are not considered in this study [8], [9], [36], [37].

Analog Processing In-DRAM: This category employs charge sharing among multiple activated DRAM cells for bitwise processing [3], [31]. Using charge-sharing, an analog majority function is performed, where the charge of the majority of cells determines the value read on the SA. In [31], concurrent triple-row activation (TRA) is presented to enable bitwise AND/OR operations using the majority function. The NOT function is achieved using additional dual-contact cell (DCC) rows [31]. In [3], [29], a bit-serial adder utilizing a 5-bit analog majority function is implemented, and operands and operation results are read and stored in a column-based organization to avoid shift operations between bitlines. However, this requires a costly transpose operation to write/read data to/from DRAM.

The main and most critical limitation of analog PIM is its susceptibility to process variation, ambient and internal noises, and cell charge leakage [25]. This is because the perturbation signal (Vs) during multiple-cell activations, is smaller than a single cell in the worst-case scenario, when the numbers of '0's

and '1's stored in the activated cells are close to each other. Another challenge is that the initial operand data are destroyed by performing the PIM operation, necessitating the use of multiple bulk copies [17]. Furthermore, the operation result is stored in multiple cells, resulting in extra energy dissipation.

Semi-Digital Processing In-DRAM: Semi-digital PIM approaches activate only a single wordline at a time. In [10], AND and OR operations are achieved using off-the-shelf DRAMs by sequentially activating operand rows at a high frequency, simulating simultaneous activation of multiple rows. However, this technique does not support NOT operation, rendering complete logic realization infeasible. Moreover, controlling the timing of wordline voltage and the high-frequency voltage swing on the wordlines is challenging, causing increased leakage current and thus compromising reliability goals. [14].

To overcome these, a technique presented in a [34] eliminates extra bulk copy and error-prone charge sharing between multiple cells by utilizing additional voltage control states in the precharge circuit and the SA. This technique does not require reserved cell rows for operation, but rather drives the sense amplifier pull-down circuit with a voltage of half-VDD instead of GND during the operation. However, this significantly reduces the SA drive current and stability by approaching the threshold voltage of the transistors to VDD, which affects overall reliability. Additionally, the NOT operation is performed using extra DCC rows.

Another work [33] leverages the characteristics of a diode to implement logic operations. In this technique, two extra DRAM rows are utilized, where the cells in each column are connected together with a diode to perform OR and AND operations. This approach is faster than previous techniques, the charge of the cell containing the operation result is degraded by the threshold voltage drop on the diode, which significantly reduces its reliability. One of the advantages of semi-digital PIM over analog PIM is avoiding error-prone charge sharing between cells. However, as shown in this study, these methods still severely hurt reliability. Moreover, they require more modifications to the DRAM structure for the desired functionality.

The most relevant previous methods include Ambit [31], ROC [33], ELP2IM [34], PIPF-DRAM [28], and Lacc [8].

Ambit employs charge sharing among multiple cells containing operands by connecting them simultaneously, and utilizes the final developed voltage (V_S) on the bitline to perform a majority function. The majority function is then used to perform more complex bitwise operations. In Ambit, additional DRAM cell rows are considered and activated simultaneously, with operands copied to these rows before the operation begins.

The key idea of ROC is to leverage the diode-like behavior of an NMOS transistor with the gate connected to its sources, allowing for cell charging while preventing discharge in the opposite direction. This technique employs basic bitwise operations such as NOT, AND, and OR to perform more complex operations like XOR.

ELP2IM and PIPF-DRAM, on the other hand, utilize the remaining charge on the bitlines from the previous cell access to perform bitwise operations. PIPF-DRAM is based on a special DRAM structure called PF-DRAM [27], which eliminates the precharge phase and modifies the controlling signals to enable bitwise operations. Lacc [8] is another processing-in-DRAM technique aiming at CNN acceleration which performs X(N)OR operations as well. However, this technique is not considered in our evaluations due to its bitwise computing abilities being limited to X(N)OR.

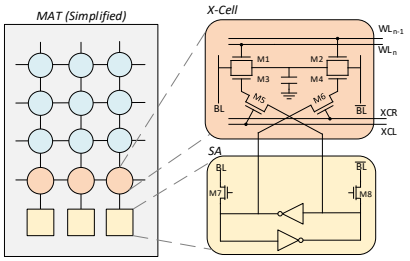


Fig. 2: Structure of WISEDRAM. Precharge circuitry and SA boosters are omitted for readability.

III. Proposed PIM Technique

In this section we describe the proposed technique for processing in memory, called WISEDRAM (a Reliable Bitwise In-DRAM Accelerator). Then we discuss different bitwise operations that WISEDRAM can perform.

A. WISEDRAM Structure

WISEDRAM integrates a row of special cells, called X-Cells, into conventional DRAM for bitwise operations, as shown in Figure 2. X-Cells resemble dual-contact cells [18], [31], with four added transistors (*M3* to *M6*) that allow access from either bitline based on the previous SA read.

By activating the XCL (X-Cell Connect Left) and XCR (X-Cell Connect Right) control signals, the X-Cell induces a voltage perturbation on either the left or right bitline, depending on whether the previous value, from the previous cell access, on the SA is '0' or '1', respectively. This ability can be leveraged to perform bitwise operations. While the provided examples focus on operands connected to BLL (Bitline Left) for clarity, it's worth noting that the operands can be on any MAT cell, thanks to the X-Cell's dual-contact nature.

WISEDRAM retains the same structure and memory cell access operations as conventional DRAM. It is important to note that the SA structure was not modified, unlike many of the previously proposed techniques discussed in Section IV. In conventional DRAMs, decoupler transistors (*M7* and *M8*) are controlled by the DCL (Bitline Decoupler) signal to enhance memory access latency and enable the sharing of SAs between adjacent MATs. In the DDR5 architecture, the activation of a row of 64 Kb during each cell access enables the parallel operation of 64 Kb through the use of WISEDRAM. This technique leverages the substantial internal bandwidth of DRAM to achieve significant acceleration of bitwise operations.

B. X(N)OR in WISEDRAM

Due to the significance of X(N)OR as a major bitwise operation, WISEDRAM structure has been optimized for efficient X(N)OR operations. The rationale behind WISEDRAM's design is based on the XOR operation between two operands, *A* and *B* ($A \oplus B$).

In order to compute $A \oplus B$ serially, the value of one operand (here *B*) is observed. If $B = 0$, it can be seen that $A \oplus B = A$. Otherwise, $A \oplus B = \overline{A}$. This view of XOR as a "controlled NOT" is the fundamental operation of WISEDRAM.

The computation steps for XOR in WISEDRAM are illustrated in Figure 3, and Figure 4 shows the respective waveforms, extracted from simulations explained in Section IV.

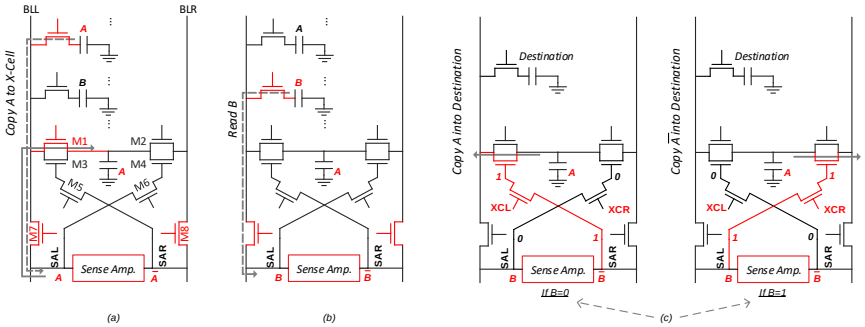


Fig. 3: XOR Computation in WISEDRAM: (a) Copying *A* to X-Cell, (b) Reading *B*, (c) Accessing X-Cell via XCL and XCR to copy X-Cell to Destination. For XNOR operation, *A* must be copied from *M2* in the first step, thus storing the complement of *A* in the X-Cell.

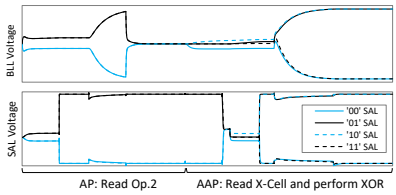


Fig. 4: Waveform of BLL and SAL in WISEDRAM while computing XOR.

To begin, one of the operands (in this case, *A*) is copied to the X-Cell using a row copy as introduced in RowClone [30]. This is achieved through an overlapped Activate-Activate-Precharge (AAP) primitive [31], which can be observed in Figure 3 (a). The X-Cell is a dual-contact cell that features access transistors connected to either bitlines in each pair. With this structure, operands can be stored on any cell in the memory array.

In the second step, as shown in Figure 3 (b), the second operand (*B*) is read, causing its value to appear on the SA nodes. The left SA node, SAL, stabilizes at the value of *B*, while the right node, SAR, takes on the complement value ($\overline{B}$). Once *B* has been restored, the bitlines are precharged and equalized to *VDD/2*, the same as conventional DRAM cell access. This step is an Activate-Precharge (AP) primitive.

In the final step of X(N)OR operation, the control signals XCR and XCL activate *M5* and *M6*, as shown in Figure 3 (c). If $B = 0$, the voltage on SAR activates *M3*, while *M4* remains inactive. In this case, the X-Cell is accessed from the BLL, and the voltage perturbation of *A* appears on BLL. Alternatively, if $B = 1$, the value of SAL activates *M4*, and *M3* remains inactive. In this case, the X-Cell is accessed from the BLR (Bitline Right), and the voltage perturbation of *A* appears on BLR. After accessing the X-Cell, the SA is equalized and connected to the bitlines for reading the value, through *M7* and *M8*. The read value is A if $B = 0$, and $\overline{A}$ if $B = 1$. The wordline for the target cell is also activated during this step, and this activation is overlapped [30], making the third step an AAP primitive. This step is similar to a row copy, but instead of activating the source (X-Cell) using a wordline, it is activated using XCR and XCL signals. As a result, the value $A \oplus B$ is written back to the selected destination operational cell.

The computation of XNOR is similar to XOR. Eq. 1 serves as the basis for XNOR computation in WISEDRAM. Since the X-Cell is dual-contact cell, instead of copying A in the first step (see Figure 3 (a)), $\overline{A}$ can be copied in X-Cell. By continuing the steps as outlined in Figure 3, the resulting value stored in the destination cell will be $\overline{A} \oplus B$, which corresponds to the XNOR of *A* and *B*.

$$\overline{A \oplus B} = \overline{A} \oplus B \tag{1}$$

The procedure for performing the X(N)OR operation indicates that it can be executed in three DRAM AP/AAP primitives (one AP and two AAP). This is almost as fast as the theoretical optimal number of operations required (three AP primitives), since an access to two operands and one destination is inevitable.

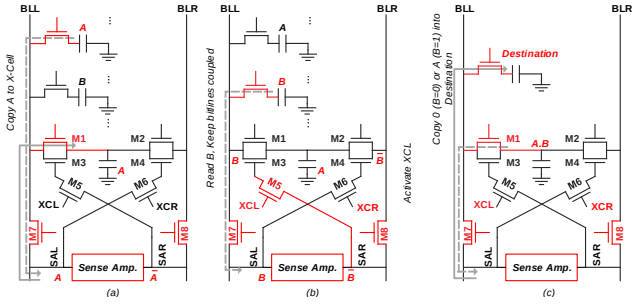


Fig. 5: AND Computation in WISEDRAM: (a) Copying *A* to X-Cell, (b) Reading *B* and activating XCL, (c) Copying X-Cell to Destination. For NAND, the complement of the X-Cell can be copied into the destination cell.

C. (N)AND and (N)OR in WISEDRAM

This subsection explores how WISEDRAM's X-Cell can be leveraged to perform (N)AND and (N)OR operations in three DRAM AP/AAP primitives (one AP and two AAP), which is almost as fast as the theoretical minimum time required to compute these operations (three AP primitives).

1) (N)AND Computation: The computation of (N)AND follows a similar procedure to XOR, but with a distinct application of the X-Cell. The logical operation of AND between two operands, represented as $A.B$, can be formulated using Eq. 2. The result of the operation is '0' if the first operand is '0', irrespective of the second operand. On the other hand, if the first operand is '1', the output is equivalent to the second operand. This principle underlies the computation of AND in WISEDRAM, as depicted in Figure 5.

$$A.B = \begin{cases} 0 & B = 0 \\ A & B = 1 \end{cases} \tag{2}$$

Initially, the first operand (designated as A) is copied into the X-Cell. Next, the second operand (designated as B) is read, while the sense amplifier remains coupled and the bitlines are not pre-charged. At this point, the sense amplifier drives the bitlines BLL and BLR with the values of B and its complement $\overline{B}$, respectively. Simultaneously, the XCL signal is activated. If $B =$'0', then the value on the SAR, which is the complement of B, will be equal to '1'. As a result, *M3* will be activated, and the value on the X-Cell will be overwritten by the value of BLL, which is equivalent to '0' in this case. Conversely, if $B =$'1', the value stored in the X-Cell (A) will remain unchanged and X-Cell contains *A.B*.

The final step is to copy the X-Cell into the destination cell using an AAP primitive. A regular row copy will save

AND output into the desired destination row. The second access transistor (*M2*) of X-Cell can be leveraged to copy the complement of computed value into the destination, which is equal to $\overline{A.B}$, NAND operation.

2) (N)OR Computation: Computing (N)OR is akin to (N)AND, with minor differences. If XCR is connected in the second step instead of XCL, BLR will overwrite the X-Cell with '0' when $B =$'1'. This will result in the computation of $A.\overline{B}$. The first step can also be changed to copy $\overline{A}$ to the X-Cell, using its dual-contact feature. This results in the computation of $\overline{A.\overline{B}}$ which is equal to $\overline{A} + B$, NOR of the operands is stored in the X-Cell. Copying X-Cell or its complement into the destination fulfills the computation of NOR or OR, respectively. Thus, (N)OR can also be computed in three AP/AAP.

D. Copy and NOT in WISEDRAM

Row copy functionality is present in conventional DRAM structures, which has been studied in previous literature such as [30]. This operation is utilized for various computations, as demonstrated in the preceding subsections and other research works [28], [31], [33].

Since the X-Cell is inherently dual-contact, it can also be used to invert operands during a copy operation to perform the NOT operation similar to the method described in [31].

E. General-Purpose Computing

Previous work has made the case for more advanced operations in DRAM arrays [3], [13]. Since DRAM bitlines are inherently isolated from eachother, these techniques use bit-serial operations, wherein operands are stored vertically in one bitline. By doing so, a bitwise in-memory approach will be able to compute more advanced operations such as addition and multiplication. This work serves as a drop-in replacement for the circuits that are used in [3], [13]; therefore, the same approach can be leveraged to perform non-bitwise computation on WISEDRAM and the circuit-level improvements will propagate to upper layers.

IV. Experimental Evaluations

In this section, we conduct a comparative analysis between WISEDRAM and the state-of-the-art in-DRAM processing techniques, including Ambit [31], ROC [33], ELP2IM [34], and PIPF-DRAM [28].

For circuit-level evaluation of functionality, performance, robustness, and energy consumption, we employ a 16 nm multi-gate technology model in HSPICE, under the same evaluation conditions. Simulation parameters are summarized in Table I.

A. Functionality and Performance

To assess the latency of bitwise operations in comparison with other state-of-the-art techniques, we implemented PIPF-DRAM, Ambit, ROC, and ELP2IM with all of the optimizations presented in their respective works to enhance latency, and

TABLE I: Circuit-level simulation parameters.

Technology	16 nm PTM-MG
VDD/VPP	1.1 V/1.8 V [1]
MAT size	512×512 cell
C_{Cell}/C_{BL}	24 fF/144 fF [7], [27]
Transistors W/L	SA 10/1, decoupling 2/1, access 1/1, precharge 2/1 [11], M1-M4 1/1, M5-M6 2/1 (Figure 2)
Process Variation	Gaussian distribution: σ = 2.5%, 5%, 7.5%, 10%, 12.5%, and 15% m = nominal value for V_{th}, capacitances, and resistances

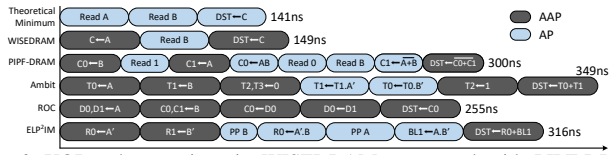


Fig. 6: XOR sub-operations in WISEDRAM, compared with PIPF-DRAM, Ambit, ROC and ELP2IM and the theoretical minimum. In ELP2IM, PP indicates pseudo-precharge.

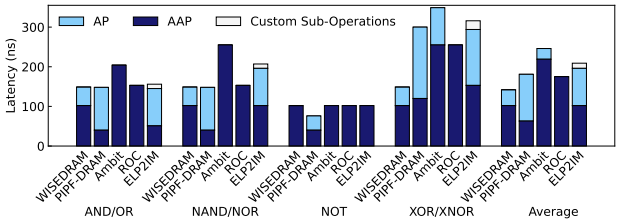


Fig. 7: Latency of operations in WISEDRAM, compared with PIPF-DRAM, Ambit, ROC, and ELP2IM.

then compared their latency with that of WISEDRAM. Figure 6 shows the breakdown of sub-operations to perform XOR in these PIM techniques, alongside the minimum latency that can be theoretically achieved in a PIM technique where the result can be stored in an arbitrary row. Since accessing two operands is inevitable and they must both be restored and maintained, two separate AP primitives are necessary, and another is required to store the result in an arbitrary location, which adds up to three AP operations for a complete XOR computation. WISEDRAM is the closest to this minimum, and requires two AAP primitives and one AP primitive to compute an XOR, which takes 149ns and is only 8ns slower than the absolute minimum.

Figure 7 compares the latency of all bitwise operations. As shown in the figure, WISEDRAM outperforms PIPF-DRAM, Ambit, ROC, and ELP2IM by 101%, 134%, 71%, and 112% for executing X(N)OR, respectively. The AAP operation performs a copy operation between rows, while the AP operation performs cell access. ELP2IM uses some custom sub-operations, other than activation and precharge, to control its special sense amplifier structure to precharge bitlines in an imbalanced manner, which adds extra delay to execute bitwise operations. On average, WISEDRAM exhibits lower latencies compared to PIPF-DRAM, Ambit, ROC, and ELP2IM in executing bitwise operations, with reductions of 27%, 72%, 22%, and 46%, respectively.

B. Energy Consumption

In the execution of operations within the memory structure, the primary contributor to energy consumption is the charge/discharge of high-parasitic capacitance bitlines [28]. This energy dissipation occurs in the sense amplifier pull-up transistors and bitlines precharge circuitry, as illustrated in Figure 1. Another significant energy-intensive operation is the activation of wordlines, which results in energy dissipation within the local address decoders and drivers. WISEDRAM and ROC can perform memory cell access using the same technique as the PF-DRAM structure during PIM operation, similar to PIPF-DRAM. The PF-DRAM structure eliminates the bitline precharge procedure from the cell access process, resulting in a significant reduction in energy consumption and delay during cell access. We evaluated WISEDRAM and ROC on conventional and PF-DRAM structures for fair assessment.

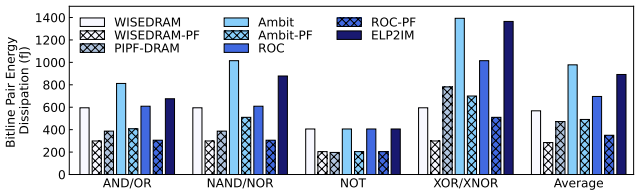


Fig. 8: Energy dissipation on a biline pair using different PIM techniques.

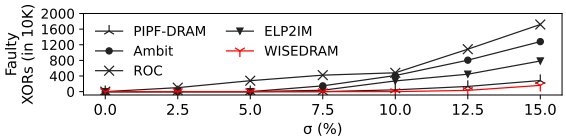


Fig. 9: Process variation effect on faulty bitwise operation.

In conventional DRAM, the AP and AAP operations consume 189 fJ and 203 fJ, respectively. However, in the case of PF-DRAM structure, their energy consumption is reduced to 95 fJ and 102 fJ, respectively. The dissipated energy for bitwise operations on a single bitline using different in-memory techniques is presented in Figure 8. In WISEDRAM, XOR and XNOR operations on a single bit consume 595 fJ, making it 31%, 134%, 70%, and 129% more energy-efficient than PIPF-DRAM, Ambit, ROC, and ELP2IM, respectively. Moreover, by utilizing the PF-DRAM memory access procedure in WISEDRAM, these values are hugely improved to 1.61×, 3.65×, 2.39×, and 3.56×, respectively.

WISEDRAM, equipped with PF-DRAM cell access technique, achieves energy efficiency gains of 29% and 161% over PIPF-DRAM, the most energy-efficient PIM technique among the previously evaluated methods, for the execution of (N)AND/(N)OR, and XOR/XNOR operations, respectively. On average, WISEDRAM (PF) is 99%, 65%, 388%, 72%, 144%, 22% and 212% more power efficient than WISEDRAM, PIPF-DRAM, Ambit, Ambit (PF), ROC, ROC (PF), and ELP2IM, respectively, indicating its superior energy efficiency.

C. Robustness

For an in-memory processing technique to achieve an acceptable yield rate, it must be robust against process variation. To evaluate this, we conducted Monte Carlo simulations using HSPICE, with 10 K rounds of simulations. The operational DRAM cells and sense amplifier transistors' threshold voltages, as well as the parasitic capacitance and resistance of bitlines, were randomly updated in each simulation round using a Gaussian distribution. The standard deviation of the distribution model (σ) was considered to be between 2.5% to 15% of the mean value of each parameter (m).

Figure 9 presents the fault rate versus the standard deviation of variations in parameters for various PIM techniques based on DRAM architecture. Among the evaluated state-of-the-art techniques, ROC shows the highest sensitivity to process variation. Increasing σ from 10% to 15% results in a sharp elevation of fault rate from 482 faults per 10K evaluation rounds to over 1700 faults, respectively. This susceptibility of ROC is attributed to the voltage drop on the diode due to the transistors' threshold voltage, which is ROC's key element for computation.

The next two most susceptible PIM techniques to process variation are Ambit and ELP2IM. Monte Carlo simulations show no faults for these techniques when σ is increased to 5%. However, by increasing σ to 15%, the fault rate increases to 12.8% and 7.84%, respectively.

On the other hand, PIPF-DRAM is the most robust state-of-the-art technique in the literature, with a fault rate of only 284 faults in 10 K simulation rounds at σ=15%. This robustness is attributed to the technique's ability to maintain the perturbation signal during sub-operations. However, PIPF-DRAM employs precisely unbalanced sense amplifiers to perform its operations, which can be easily affected by process variation. WISEDRAM is fault-free even when increasing σ up to 10%. In comparison with ROC, Ambit, ELP2IM, and PIPF-DRAM, the in-memory processing (PIM) technique of WISEDRAM is 9.7×, 6.9×, 3.9×, and 77% more robust against process variation, respectively, at σ=15%. The robustness of WISEDRAM against process variation can be attributed to three main factors. First, WISEDRAM employs a reduced number of sub-operations to perform an operation, thereby reducing the overall sensitivity to process variation-related failures. Second, WISEDRAM uses the same cell access procedure as conventional DRAM, making it less susceptible to process variation. Finally, conventional DRAMs use increased voltages in the sense amplifier [20] to reduce latency. This alleviates the threshold-induced voltage drop in the X-cell's transistors, which caused many errors in ROC.

D. Layout and Area Overhead

The area overhead of WISEDRAM is limited to four extra wordlines per MAT to control X-Cell operation, plus a dual-contact cell and four extra NMOS transistors per bitline pair. As shown in [31], a dual-contact cell occupies around two memory cells. The other controlling transistors, *M3* to *M4* can be fabricated with the smallest size in the technology, the same as cell access transistors, and *M5* and *M6* are slightly larger. The X-Cell can be placed near the sense amplifier to ease layout design, since folded-bitline sense amplifiers are four bitlines wide [16], [32]. The total area overhead of an X-Cell is approximately 8 memory cells per bitline pair. In a 512-cell bitline, this results in an area overhead of roughly 1.6%.

V. Conclusions

Given the maturity, capacity, and large internal bandwidth of DRAM, it is an excellent candidate to serve as a platform for Processing In-Memory (PIM). We present a novel PIM technique called WISEDRAM, which is capable of performing a range of operations including X(N)OR, (N)AND, (N)OR, NOT, and Copy. WISEDRAM can also perform an unlimited number of bitwise operations, providing greater power efficiency and reduced latency. Compared to existing PIM techniques, WISEDRAM executes X(N)OR operations, one of the most complex PIM operations, 71% faster. Additionally, WISEDRAM is 77% more robust against process variation compared to the most robust PIM technique, with only a negligible area overhead of roughly 1.6%.

Acknowledgment

This work is part of the project PID2023-146511NB-I00 funded by the Spanish Ministry of Science, Innovation and Universities MICIU /AEI /10.13039/501100011033 and EU ERDF. This publication is promoted by the Barcelona Zettascale Laboratory, backed by the Ministry for Digital Transformation and of Public Services, within the framework of the Recovery, Transformation, and Resilience Plan – funded by the European Union – NextGenerationEU.

REFERENCES

[1] "Ddr5 sdram," https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr5/16gb_ddr5_sdram_diereva.pdf, 2021.

[2] K. Al-Hawaj, O. Afuye, S. Agwa, A. Apsel, and C. Batten, "Towards a reconfigurable bit-serial/bit-parallel vector accelerator using in-situ processing-in-sram," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[3] M. F. Ali, A. Jaiswal, and K. Roy, "In-memory low-cost bit-serial addition using commodity dram technology," *Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 155–165, 2019.

[4] T. Andrulis, J. S. Emer, and V. Sze, "Raella: Reforming the arithmetic for efficient, low-resolution, and low-loss analog pim: No retraining required!" in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–16.

[5] S. Angizi and D. Fan, "Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations," in *International Conference on Computer-Aided Design*. IEEE, 2019, pp. 1–8.

[6] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 316–331.

[7] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, pp. 1–42, 2017.

[8] Q. Deng, Y. Zhang, M. Zhang, and J. Yang, "Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[9] J. D. Ferreira, G. Falcao, J. Gómez-Luna, M. Alser, L. Orosa, M. Sadrosadati, J. S. Kim, G. F. Oliveira, T. Shahroodi, A. Nori, and O. Mutlu, "pluto: Enabling massively parallel computation in dram via lookup tables," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 900–919.

[10] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "Computedram: In-memory compute using off-the-shelf drams," in *Proceedings of the 52nd annual international symposium on microarchitecture*, 2019, pp. 100–113.

[11] J. C. Gealow, "Impact of processing technology on dram sense amplifier design," Ph.D. dissertation, Massachusetts Institute of Technology, 1990.

[12] N. M. Ghiasi, N. Vijaykumar, G. F. Oliveira, L. Orosa, I. Fernandez, M. Sadrosadati, K. Kanellopoulos, N. Hajinazar, J. G. Luna, and O. Mutlu, "Alp: Alleviating cpu-memory data movement overheads in memory-centric systems," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[13] N. Hajinazar ,*et al.*, "Simdram: a framework for bit-serial simd processing using dram," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 329–345.

[14] K. Itoh, *VLSI memory chip design*. Springer Science & Business Media, 2013, vol. 5.

[15] L. Jiang, M. Kim, W. Wen, and D. Wang, "Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[16] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM circuit design: fundamental and high-speed topics*. John Wiley & Sons, 2007, vol. 13.

[17] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 288–301.

[18] S.-L. Lu, Y.-C. Lin, and C.-L. Yang, "Improving dram latency with dynamic asymmetric subarray," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 255–266.

[19] H. Luo, T. Shahroodi, H. Hassan, M. Patel, A. G. Yağlıkçı, L. Orosa, J. Park, and O. Mutlu, "Clr-dram: A low-cost dram architecture enabling dynamic capacity-latency trade-off," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 666–679.

[20] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "Rega: Scalable rowhammer mitigation with refresh-generating activations," in *44rd IEEE Symposium on Security and Privacy (SP 2023)*. IEEE, 2023.

[21] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Enabling practical processing in and near memory for data-intensive computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–4.

[22] A. Olgun, J. G. Luna, K. Kanellopoulos, B. Salami, H. Hassan, O. Ergin, and O. Mutlu, "Pidram: A holistic end-to-end fpga-based framework for processing-in-dram," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 1, pp. 1–31, 2022.

[23] G. F. Oliveira, J. Gómez-Luna, S. Ghose, A. Boroumand, and O. Mutlu, "Accelerating neural network inference with processing-in-dram: From the edge to the cloud," *IEEE Micro*, vol. 42, no. 6, pp. 25–38, 2022.

[24] G. F. Oliveira, A. Olgun, A. G. Yağlıkçı, F. N. Bostancı, J. Gómez-Luna, S. Ghose, and O. Mutlu, "Mimdram: An end-to-end processing-using-dram system for high-throughput, energy-efficient and programmer-transparent multiple-instruction multiple-data computing," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 186–203.

[25] L. Orosa, Y. Wang, M. Sadrosadati, J. S. Kim, M. Patel, I. Puddu, H. Luo, K. Razavi, J. Gómez-Luna, H. Hassan, N. Mansouri-Ghiasi, S. Ghose, and O. Mutlu, "Codic: A low-cost substrate for enabling custom in-dram functionalities and optimizations," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 484–497.

[26] J. Park, R. Azizi, G. F. Oliveira, M. Sadrosadati, R. Nadig, D. Novo, J. Gómez-Luna, M. Kim, and O. Mutlu, "Flash-cosmos: In-flash bulk bitwise operations using inherent computation capability of nand flash memory," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 937–955.

[27] N. Rohbani, S. Darabi, and H. Sarbazi-Azad, "Pf-dram: a precharge-free dram structure," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 126–138.

[28] N. Rohbani, M. A. Soleimani, and H. Sarbazi-Azad, "Pipf-dram: Processing in precharge-free dram," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1075–1080.

[29] S. Roy, M. Ali, and A. Raghunathan, "Pim-dram: Accelerating machine learning workloads using processing in commodity dram," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 701–710, 2021.

[30] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 185–197.

[31] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 273–287.

[32] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[33] X. Xin, Y. Zhang, and J. Yang, "Roc: Dram-based processing with reduced operation cycles," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[34] ——, "Elp2im: Efficient and low power bitwise operation processing in dram," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 303–314.

[35] R. Zhou, A. Roohi, D. Misra, and S. Angizi, "Flexidram: A flexible in-dram framework to enable parallel general-purpose computation," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2022, pp. 1–6.

[36] ——, "Red-lut: Reconfigurable in-dram luts enabling massive parallel computation," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022.

[37] R. Zhou, S. Tabrizchi, A. Roohi, and S. Angizi, "Lt-pim: An lut-based processing-in-dram architecture with rowhammer self-tracking," *IEEE Computer Architecture Letters*, vol. 21, no. 2, pp. 141–144, 2022.

Pipirima: Predicting Patterns in Sparsity to Accelerate Matrix Algebra

Ubaid Bakhtiar, Donghyeon Joo, and Bahar Asgari
University of Maryland, College Park
{ubaidb, dhjoo98, bahar}@umd.edu

***Abstract*—While sparsity, a feature of data in many applications, provides optimization opportunities such as reducing unnecessary computations, data transfers, and storage, it causes several challenges, too. For instance, even in state-of-the-art sparse accelerators, sparsity can result in load imbalance; a performance bottleneck. To solve such challenges, our key insight is that if while reading/streaming compressed sparse matrices we can quickly anticipate the locations of the non-zero values in a sparse matrix, we can leverage this knowledge to accelerate processing sparse matrices. To enable this, we propose Pipirima, a lightweight prediction-based sparse accelerator. Inspired by traditional branch predictors, Pipirima uses resource-friendly simple counters to predict the patterns of non-zero values in the sparse matrices. We evaluate Pipirima based on sparse matrix vector multiplication (SpMV) and sparse matrix-dense matrix multiplication (SpMM) kernels on CSR compressed matrices derived from both scientific computing and transformer models. On average, our experiments show $6\times$ and $4\times$ speed up over Tensaurus for SpMM and SpMV, respectively on SuiteSparse workload. Pipirima also shows $40\times$ speed up over ExTensor for SpMM. We achieve $8.3\times$, $48.2\times$ over Tensaurus and ExTensor in lesser sparse transformer workloads. Piprima consumes $5.621mm^2$ area and $544.93mW$ power using 45nm technology with predictor related components as the least expensive ones.**

I. Introduction

Over the years, sparse matrices have become crucial in various fields such as scientific computing, bioinformatics, genomics, computer vision, transformers and recommendation systems. While sparsity offers opportunities to eliminate unnecessary computations, storage, and data movement, the effectiveness of handling sparsity depends on several factors including compression methods [1] such as the frequently-used compressed sparse row (CSR) or more specific formats such as diagonal (DIA) [26], block CSR/CSC [33], list of lists (LIL) [27], and Ellpack (ELL) [15] that suit specific sparsity patterns or certain implementations. For brevity, this work targets CSR format as it stands out as the widely adopted format. However, CSR also comes with some limitations as we explain throughout this paper.

To efficiently run sparse problems, several domain-specific architectures (DSAs) [3], [9], [18], [24], [25], [29], [32], [37] have been developed, targeting sparsity in various domains such as scientific computing [2], [4], [5], [8], [10], [14], [17], [23], [28] DNNs [7], [13], [19]–[21], [34], [36] and sparse attention mechanisms [22], [35], some focusing on very specific aspects such as dataflow architectures, or implementing sparse DNNs on dense systolic arrays [11], [16]. In addition to the *domain-specific* studies, *kernel-specific* accelerators have been proposed to target common sparse matrix algebra, such as sparse matrix-vector multiplication (SpMV), sparse matrix-matrix multiplication (SpMM) sparse-sparse matrix multiplication (SpGEMM) that offer methods ranging from new microarchitectural supports for sparsity to enhancing memory bandwidth utilization. ExTensor [12], for instance, proposes a hierarchical elimination of computation in sparse environments by identifying multiplication cases where both operands are non-zero, thus avoiding unnecessary data transfers. Some of the prior studies integrate compression format into their co-optimization cycle. For instance, Tensaurus [30] employs a novel format for accessing sparse data in a vectorized and streaming manner, optimizing memory bandwidth usage by splitting dense vectors into tiles. The exploration of recent sparse accelerators underscores two major challenges. First, despite structured sparsity in matrices, these accelerators often perform standard operations without leveraging these patterns for quicker computations. Second, fine-grained load imbalance, as a result of unstructured sparsity, hinders the full utilization of parallelism capabilities as also discussed in [31].

To overcome these challenges with minimal hardware and latency overhead and without relying on extensive preprocessing, we introduce Pipirima[1], a novel approach using a lightweight, rapid sparsity pattern predictor based on counter-based prediction (CBP), similar to branch predictors. Pipirima uses a 1-bit counter to identify matrix structural patterns, enabling quick identification and omission of standard operations. In non-diagonal matrices, this prediction method forecasts the number of non-zero (NNZ) values per row in the streaming blocks to improve load balance and enhance fine-grained parallelism. Unlike traditional CSR-based computation, Pipirima distributes values based on the state of the predictor, and includes a straightforward inspector for the verification process for prediction accuracy and tracking the states. We also propose a novel architecture with Sigma-like [24] adder tree and arrangement of multipliers that uses the predictions to do the SpMM and SpMV operations. We evaluate Pipirima using a cycle-accurate simulator, conducting experiments on the CSR-based random and diagonal matrices from SuiteSparse dataset [6], transformer models, and synthetic random matrices. Our experiments demonstrated a significant speed up compared to baseline sparse accelerators; ExTensor [12] and Tensaurus [30] at the expense of very small memory and computational overhead of CBP predictors and Prediction Inspectors. The area and power consumption of

[1]Pipirima is a star in the zodiac constellation of Scorpius.

Pipirma are also very small, with Prediction Inspectors as the least expensive components.

II. Motivation & Key Insight

Challenges & Acceleration Opportunities: We explore the challenges and opportunities of recent SpMM accelerators from two angles. ***(1)*** *The first issue is that even if the sparse matrices manifest structured sparsity, such as diagonal, the SpMM accelerators still undergo standard operations instead of taking shortcuts to quicker results.* ***(2)*** *The second issue concerns the fine-grained load imbalance, referring to the uneven distribution of NNZ values among units of on-chip memory and computational units.* This imbalance prevents sparse matrix algebra from maximizing fine-grained parallel capabilities of hardware. Our key insight to solve the aforementioned challenges is to use simple prediction mechanisms to first predict a categorical pattern of sparsity in a matrix and accordingly skip some computations and second, if the matrix does not have a pattern, use prediction to enable ideal load balancing. In the following we provide more details about our key insights to solve each of the challenges.

1) ***D/R Matrix Predictor:*** Given the structural peculiarity of diagonal matrices and since the indices of NNZ values in a diagonal matrix are known, we can bypass some steps of processing a sparse matrix (e.g., reading offsets and column indices when CSR format is used) and directly feed the input values into the compute unit.

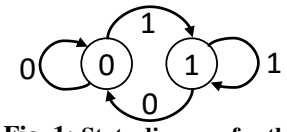


Fig. 1: State diagram for the D/R Matrix Predictor– States 0 and 1 indicate diagonal and random.

To quickly identify the structure of a matrix blocks, we use a single bit for prediction. As shown in Figure 1, D/R Matrix Predictor includes two states: *(i) Random Matrix (ii) Diagonal Matrix*, where the prediction problem is mapped to a binary problem by representing ***0*** for random matrix and ***1*** for diagonal matrix. The predictor works by observing the structure of the blocks – the current block is predicted to be the same as the previous one.

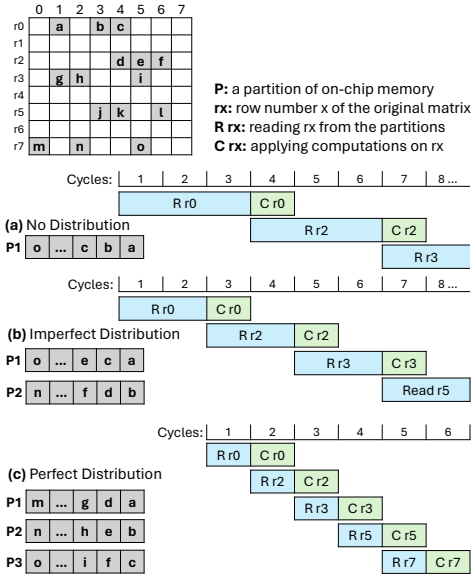


Fig. 2: Distribution matters– (a) Serial placement of NNZ values, resulting in bottleneck. **(b)** Sub-par parallelism due to NNZ distribution among two memory partitions. **(c)** Maximum parallelism due to perfect NNZ distribution.

2) ***NNZ/row Predictor:*** Before going through our solution to the second challenge, we provide more details about the challenge of fine-grained load imbalance starting with reviewing the importance in load distribution.

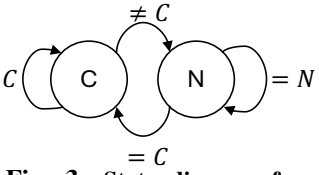
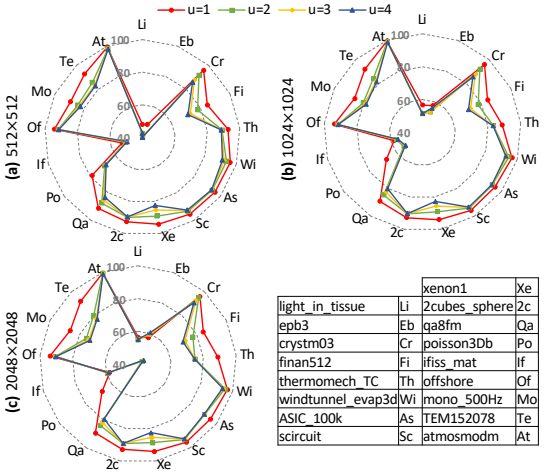


Fig. 3: State diagram for the NNZ/row Predictor– States C and N indicate the current and new number of non-zero/row.

Why distribution is important? Here, we assume that we have a streaming accelerator including a pipeline buffer. To process a sparse matrix such as the one shown in Figure 2, for each row, we need to first *read* the NNZ elements from the on-chip buffer (i.e., `R rx` in Figure 2) and then perform the *computation* of that row (i.e., `C rx` in Figure 2), where `R rx` and `C rx` are pipelined. Given such a pipeline, Figure 2 compares three cases: **(a)** having only one partition of on-chip memory, which serializes reading the NNZ values, **(b)** having more than one partition of on-chip memory but an imperfect distribution of NNZ values, which reduces the `R rx` time but does not completely solve the problem, and **(c)** a perfect distribution of NNZ values to as many partitions as they need which minimizes the `R rx` and results in a more balance pipeline. While this figure illustrates the importance of load balance given a dot-product based computation, a row-wise or outer-product based implementation suffer from a similar load-imbalance issue because of sparsity.

Our key insight to achieve the perfect distribution with minimal costs. Pipirima enables the optimized implementation shown in Figure 2c. Our key insight to enable this is to use a simple state machine shown in Figure 3 that predicts the NNZ values for the current block based on history of the previous blocks (like branch predictors). Using such a simple predictor, by just reading a single counter, as we stream values



		xenon1	Xe
light_in_tissue	Li	2cubes_sphere	2c
epb3	Eb	qa8fm	Qa
crystm03	Cr	poisson3Db	Po
finan512	Fi	ifiss_mat	If
thermomech_TC	Th	offshore	Of
windtunnel_evap3d	Wi	mono_500Hz	Mo
ASIC_100k	As	TEM152078	Te
scircuit	Sc	atmosmodm	At

Fig. 4: Row homogeneity % for the last "u" neighbours of SuiteSparse workloads

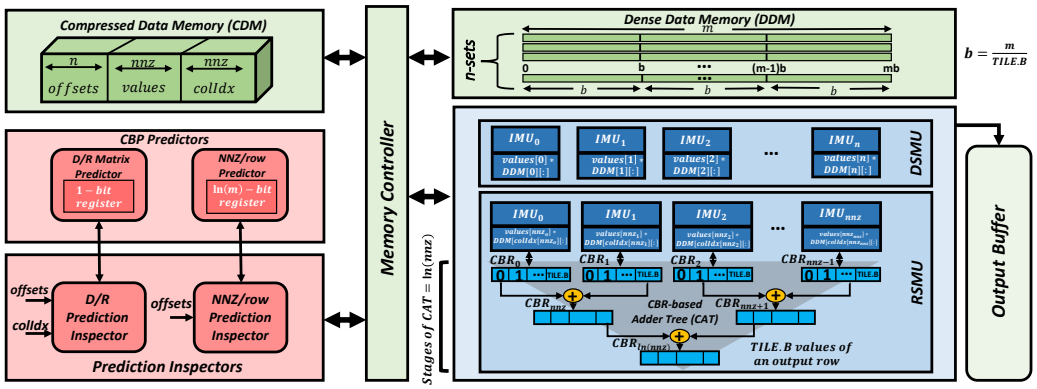


Fig. 5: Architecture of Pipirima– The memory controller sends the input data based on the predictions from CBR and forwards it to the Prediction Inspectors and Compute Units.

and indices, we distribute them evenly.

Why does the proposed solution work? In Figure 4, we run experiments on some matrices under different partition sizes from the SuiteSparse dataset to see how many rows have the same NNZ values as their last u neighbors. We see that on average 78-85% of the rows are homogeneous to their last u neighbors in terms of NNZ values. Notably, when $u = 1$, meaning only the last neighbor is considered, the resemblance is most pronounced. This finding motivates us to exploit the structural coherence of sparse matrices to ensure perfect distribution and acceleration of computations associated with sparse matrix algebra.

III. Pipirima

A. High-level Overview of Pipirima

In this work, we focus on accelerating sparse algebra computations assuming that all the essential data is available within the on-chip memory. Therefore, we do not delve deeply into data movement between off-chip memory and the on-chip network. We implement a tile-based mechanism in our compute units, which is pivotal for facilitating fine-grained parallelism in each multiplication operation, making Pipirima flexible to adapt to different configurations. (details in Section IV-B). Figure 5 shows the architecture of Pipirima, the details of which are broken into three groups: memory components, prediction components, and computational components (shown in green, red, blue in Figure 5, respectively).

B. Memory Components

1) ***Compressed Data Memory (CDM):*** Compressed Data Memory (CDM) is an on-chip memory that is responsible for holding the sparse input data compressed in CSR format. CDM has a predominantly read-oriented role in the context of Pipirima. Upon receiving a command from the memory controller, the CDM retrieves and dispatches the requested compressed data to the designated unit within the architecture.

2) ***Dense Data Memory (DDM):*** Dense Data Memory (DDM) is responsible for storing the dense matrix or vector operand. For an $n \times m$ matrix partition, DDM has n sets of m-size blocks. These blocks are split into *TILE.B* number of tiles. Dense matrix is stored in row-major format in these sets. This improves the spatial locality of a dense matrix such that when referring to a row-vector and its tile, the compute units can always go to the specific set without any memory miss.

C. Prediction Components

1) ***CBP Predictor:*** The D/R Matrix Predictor and NNZ/row Predictor reside in this unit. Each of these predictors have one register that is responsible for holding the current state. The D/R Matrix Predictor has a 1-bit register as it represents only two states. For an $n \times m$ matrix, the NNZ/row Predictor consists of a $ln(m)$-bit register, which is an insignificant hardware cost associated with our CBP predictors. The registers in CBP predictors receive an input from their corresponding prediction inspectors to update the states.

2) ***Prediction Inspector:*** To keep the states of the predictor updated, we need to verify whether a prediction was correct or not and change the state accordingly if needed. In this section, while our current emphasis lies in detailing the prediction verification steps tailored for CSR format within Pipirima, the high-level approach holds significant potential for seamless adaptation to other compressed formats. Following two prediction inspectors comprise Pipirima:

D/R Prediction Inspector– In a diagonal matrix the consecutive values in the column indices and row offset arrays have a difference of 1 and the size of these arrays is equal to n for an $n \times n$ matrix. Pipirima exploits these simple properties to inspect the predictions. The prediction verification process is performed in a pipeline with three stages; compute, compare, and output In the compute stage, the subtraction operation is done and the compare stage compares it with the D/R Matrix Prediction. If the comparison fails, this signifies a misprediction and Pipirima does the following:

- If prediction=1 (i.e., diagonal), it flushes the pipeline, resets the values sent to compute units, sets the state register to 0 (random), triggers the NNZ/row Predictor and gets the correct data from the memory for the Prediction Inspector.
- If prediction=0 (i.e., random), it flushes the pipeline, resets the values sent to the NNZ/row Predictor, sets the state register to 1 (diagonal), signals the memory controller, and sends values to the compute unit directly.

In the case of correct prediction, Pipirima continues with its normal working without any changes in the state register.

NNZ/row Prediction Inspector– The mechanism of this inspector is simple, too, as the number of non-zeros for a row in a block can be calculated easily based on the offsets. If the comparison is true, this signifies a correct prediction, hence no change in the state. A misprediction by the the NNZ/row predictor does not cause any changes in the functionality as it only defines the load distribution. Therefore, an NNZ/row misprediction only leads to a less than optimal performance and detecting such a misprediction only requires updating the state for the next predictions.

D. Computational Components

1) ***If-Mul Units (IMU)****:* This unit is responsible for multiplication but even though the dense matrix is not sparse yet there can be few zero values that result in wasted number of multiplication cycles. To avoid this inefficiency, we introduce a condition using a comparator and multiplexer, to check if the values from DDM are non-zero or not. If it is equal to zero, it is pushed to the CBR, else the multiplication units (MU) would be triggered. IMU does multiplications in parallel fashion depending on the tiling factor (*TILE.B*) of dense matrix B. IMU executes *TILE.B* number of multiplications in each step and forwards the data to the CBR, whereas the total number of steps are $b = \frac{m}{TILE.B}$. Pipirima is a versatile hardware that can be configured to implement SpMV as well as SpMM using dot product or row-wise product. For SpMV, each IMU will do one $values[nnz_i^j] \times DMM[0][colIdx[nnz_i^j]]$ operation. In the case of SpMM, each IMU does one $values[nnz_i^j] \times DMM[colIdx[nnz_i^j]][k]$ operation.

2) ***Contiguous Block Registers (CBR)****:* The partial outputs are stored in CBR, the size of which is insignificant as it only holds $TILE.B$ number of partial outputs. As shown in Figure 5, each IMU has a corresponding CBR where data is placed contiguously to improve the spatial locality of CBR for addition operations. This is for random sparse matrices only as the output of an SpMM with a diagonal matrix does not result in‘ partial outputs.

3) ***Diagonally Sparse Multiplier Unit (DSMU)****:* In case of diagonal prediction from D/R Matrix Predictor, the memory controller sends the values array data to the DSMU. In the case of diagonal sparsity, the multiplication boils down to $values[nnz_i] \times DDM[i][:]$ multiplication for each IMU. There is no need for adders in DSMU. Hence, the output of multiplications is sent directly to the output buffers.

4) ***Randomly Sparse Multiplier Unit (RSMU)****:* For random sparse matrices, there can be multiple non-zero values per row. The number of non-zero values are already known to the compute unit by the virtue of NNZ/row Predictor. Based on the prediction, RSMU reads that number of non-zero values equal to the prediction and allocate each of them to an IMU. An IMU multiplies the i^{th} non-zero of j^{th} row with the corresponding vector from DDM (i.e., $values[nnz_i^j] \times DDM[colIdx[nnz_i^j]][:]$), and stores the result in the corresponding block in CBR. For both DSMU and RSMU, we harness fine-grain parallelism by allowing each IMU to do *TILE.B* number of multiplication operations at a time. We can observe that Pipirima informed by the NNZ values per row (due to the prediction) can now employ multiple IMUs concurrently and does not have to segregate offsets load and values/indices load.

TABLE I: Pipirima configurations for SpMMs and SpMV.

Hardware Features		CFG-1 SpMM Row-wise Product	CFG-2 SpMM Dot-product	CFG-3 SpMV
TILE.B		4	4	1
RSMU	Count#	1	4	1
IMU	Count#	10	40	10
	Mul-ops/step	TILE.B=4	1	1
D/R Matrix Predictor	Register	1-bit		
NNZ/row Pr.	Register	$ln(Partition_{size})$ E.g.:	$512B \times 512B$: 9-bit $1KB \times 1KB$: 10-bit $2KB \times 2KB$: 11-bit	
CBR	Count#	19	76	19
	Size/CBR	16B	4B	4B
	Total Size	320B	320B	80B
CDM	Size	64KB		
DDM	Size	128KB		
Area (mm^2)		5.621	5.621	4.925
Power Consumption (mW)		544.93	544.93	445.42

5) ***CBR-based Adder Tree (CAT)****:* IMUs are doing $TILE.B$ number of multiplication operations in each step. These partial outputs are pushed to the corresponding CBR. We utilize a SIGMA-like [24] CBR-based Adder Tree (CAT), designed to accumulate results from adjacent CBR units. CAT runs in parallel to the IMUs, concealing the partial outputs accumulation latency. Figure 5 shows CAT inside an RSMU. CAT has $ln(nnz)$ number of stages – if a row has four non-zeros, CAT will have two adder stages. The contiguous placement of data within CBRs facilitates addition across different CBRs without the need for index matching, hence streamlining the process. The final outputs of the adder tree are $TILE.B$ elements of the j^{th} row of the output matrix and are moved to the output buffer.

IV. Experimental Setup

A. Workloads

SuiteSparse Dataset. We select a combination of random and diagonal matrices from SuiteSparse [6], few listed in Figure 4 with varying patterns of sparsity. The size of matrices range from $6M \times 6M$ to $1k \times 1k$. We break down, stream, and process these matrices into 512×512, 1024×1024, and 2048×2048 partitions.

Synthetic Matrices. To analyze the behavior of Pipirima on highly dense matrices to more precisely investigate its impact on metrics such as misprediction, we also use synthetic random matrices of size 4096×4096 with varying density.

Transformer Workloads. To evaluate Pipirima for a wider range of applications, we use Sanger's [22] code to create sparse matrices from the multi-head operations of transformer BERT trained on SQuAD v1.1. These matrices have a higher density (i.e., up to 32%) and are smaller than the SuiteSparse matrices. Therefore, here we use smaller partition sizes of 16×16, 32×32 and 64×64.

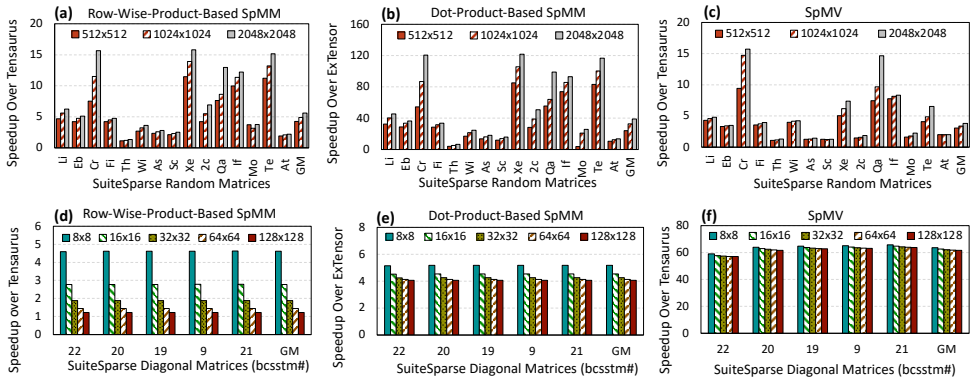


Fig. 6: Speed up for SuiteSparse matrices– (a), **(b)**, and **(c)**: Random Matrices; **(d)**, **(e)** and **(f)**: Diagonal Matrices

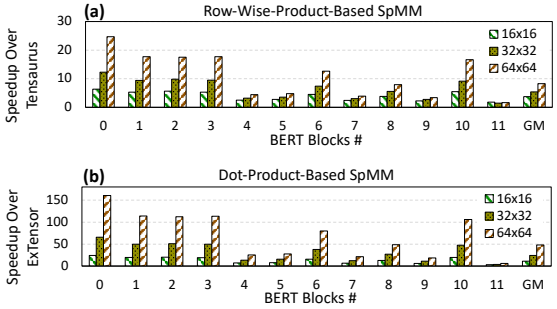


Fig. 7: Speed up for Transformer matrices– (a) row-wise product and **(b)** dot-product.

B. Baselines & Simulation Setup

We use ExTensor [12] as our baseline for dot-product-based SpMM; and Tensaurus [30] for row-wise-product-base SpMM as well as SpMV. In both of these baselines, we assume that the required data for a given computation resides in the on-chip memory. They are implemented with the same configurations as presented in their respective papers. We use a cycle-accurate simulator to model both Pipirima and baselines. We implement RSMU, DSMU, Prediction Inspector, and CBP predictor in RTL using Verilog and synthesize them using Synopsys Design Compiler in TSMC 45nm library. We use CACTI 7.0 to estimate the area, power, and latency of the memory components. We are experimenting Pipirima on SpMM using row-wise and dot product as well as SpMV kernel. Table I outlines the different configurations of Pipirima. It also shows the area and power consumption.

V. Evaluations and Results

Speed up: To examine effective acceleration, improving which is the driving motivation for Pipirima, Figure 6 represents the speed up obtained for the SuiteSparse matrices and Figure 7 depicts the speed up for the multi-attention layers of BERT. These matrices are denser than SuiteSparse matrices, giving us an opportunity to evaluate Pipirima on less sparse matrices. One of the main reasons for achieving ***performance improvements over baselines*** is the parallel computational operations corresponding to each non-zero value. *The prior knowledge of matrix partition structure – diagonal or random and NNZ values per row, allows us to distribute the multiplication operations among different IMUs– key benefit of predictions.* Pipirima also uses tiling that helps to further parallelize multiplication operations within each IMU at the expense of extra multipliers. Tensuarus also uses tiling mechanism to accelerate computations. This is the reason the speed up improvement over Tensaurus is not as much as ExTensor.

Effects of Mispredictions: Mispredictions are inherent in predictive systems and can impact system performance. Mispredictions affect the number of memory accesses and computations performed by Piprima. Figures 10 a, b, and c show the relationship between these quantities on SuiteSprase matrices with varying partition sizes. We set up a ratio to study this behavior.

A ratio less than 1 signifies more overhead of prediction components, and greater than 1 signifies more overhead in traditional decompression.

To analyze the behavior on fixed size matrix, with varying density, we are using synthetic random matrices with varying partition sizes. Figures 10 d, e, and f represent a uniform behavior between mispredictions and these overheads. These graphs also depict the same behavior.

Figure 8 investigates the relationship between the density and misprediction percentage at different partition sizes. To ensure fairness in the comparison, synthetic random matrices with fixed size and varying partition sizes have been used for this experiment. From these results, we extract two important trade-offs. First, with the increasing density, percentage mispredictions increase due to the increased probability of different NNZ values in consecutive rows and decreased row homogeneity score. Second, the percentage misprediction increases with the increasing partition sizes too. This is due to the increased number of allowed states for the NNZ/row Predictor state register, making the system vulnerable to more mispredictions.

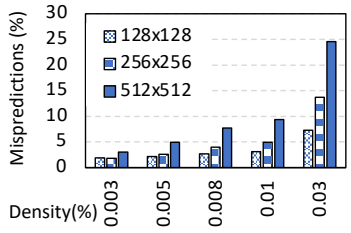


Fig. 8: Mispredictions (lower is better) vs. Density for Synthetic matrices

Overhead Analysis: CBP predictors are cheap and comprise only one register. However, the Prediction Inspectors incur memory and computational overhead. Figure 9 shows the

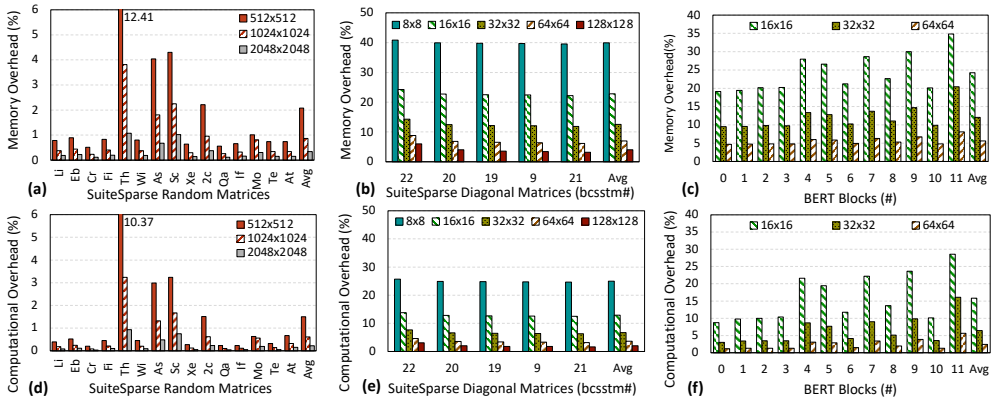


Fig. 9: Overheads of prediction inspector– (a), (b), and **(c)** Memory overhead, and **(d), (e),** and **(f)** Computational overhead

memory and computational overhead for different datasets. The overhead increases if the matrices are dense (BERT blocks) or if they are small in size. (diagonal matrices) We also observe that these overheads diminishes as the partition size increases in random matrices. The reason is fewer predictions are needed for bigger partition sizes and hence, fewer instances to be verified. For some cases (e.g. Th, Sc, As), the overhead is relatively bigger, because of their highly random distribution of non-zeros and smaller number of multiplication operations. Moreover, we observe that the Prediction Inspectors consume the least share of total area and power, that is, $0.13\%, 0.15\%$ of the total area and $0.44\%, 0.54\%$ of the total power for CFG-1, CFG-2 and CFG-3 respectively.

VI. Related Work

In recent years, several DSAs has been developed to tackle the challenges of sparse computations in various domains. Since sparsity is also prominent in DNNs, several DSAs address specific challenges in this domain, such as sparse attention mechanisms [22] and efficient execution on dense systolic arrays [11], [16]. Other DSAs atarget specific kernels icnluding SpMV, SpMM, and SpGEMM. Sigma [24], for instance, addresses irregular and unstructured SpGEMM operations and employs an adder tree network for efficient partial sum accumulation. ExTensor reduces wasted computations by identifying multiplication cases with non-zero operands ahead of time [12]. SpArch [37] optimizes the data locality of both input and output matrices in the SpMM kernel, employing a highly parallelized streaming-based merger and a condensed data representation to reduce redundant access to zero values. MatRaptor [29], a SpGEMM accelerator, uses row-wise products for high data reuse and introduces a new compression format, C^2SR. This compression format provides information about the NNZ values per row. However, load-balancing in MatRaptor still relies on round-robin allocation which does not always guarantee a perfect load balance. Another row-wise product-based study [18] proposes a counter-based matrix tiling scheme specific to CSR-compressed format to enhance parallelism in SpMM.

VII. Conclusion

This paper proposed Pipirima, a novel approach to accelerate sparse matrix algebra. By effectively predicting the structural characteristics of matrices, Pipirima enabled the tile based parallel SpMV and SpMM kernels. Extensive experimentation demonstrated Pipirima's remarkable acceleration capabilities compared to prior sparse accelerators. Importantly, Pipirima achieved these gains while maintaining minimal memory and computational overhead in its prediction components, making it a valuable asset for a wide range of applications.

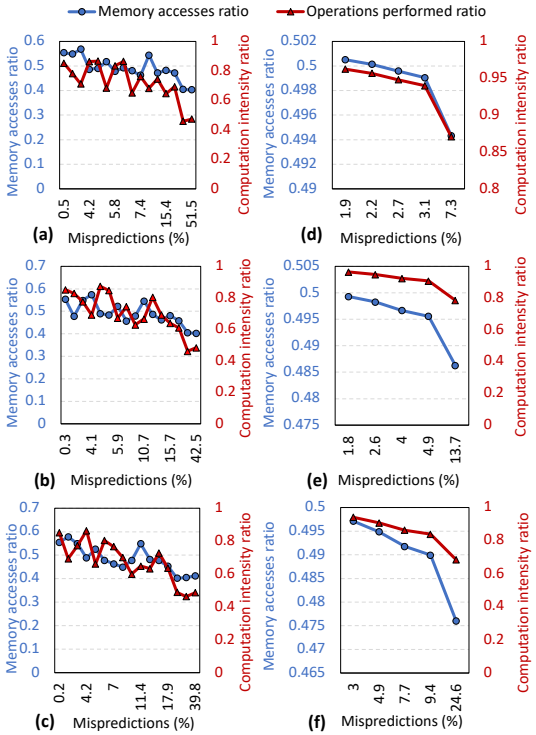


Fig. 10: Memory accesses and computation intensity ratio (higher is better) vs. Mispredictions (%)– (a), (b), and **(c)** represent the SuiteSparse non-diagonal matrices with partition sizes 512, 1024, 2048; **(d), (e),** and **(f)** represent the synthetic random matrices with partitions of 128, 256, 512 respectively.

Acknowledgments

Ubaid Bakhtiar is supported by the U.S. DoE, Office of Science, Office ACSR, the ECRP program, under Award Number DE-SC0024079. Donghyeon Joo is supported by the NSF PPoSS program, under Award Number 2316177.

REFERENCES

[1] B. Asgari, R. Hadidi, J. Dierberger, C. Steinichen, A. Marfatia, and H. Kim, "Copernicus: Characterizing the performance implications of compression formats used in sparse workloads," in *IISWC*. IEEE, 2021, pp. 1–12.

[2] B. Asgari, R. Hadidi, T. Krishna, H. Kim, and S. Yalamanchili, "Alrescha: A lightweight reconfigurable sparse-computation accelerator," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 249–260.

[3] U. Bakhtiar, H. Hosseini, and B. Asgari, "Acamar: A dynamically reconfigurable scientific computing accelerator for robust convergence and minimal resource underutilization," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 1601–1616.

[4] T. Chen, J. Botimer, T. Chou, and Z. Zhang, "An sram-based accelerator for solving partial differential equations," in *2019 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2019, pp. 1–4.

[5] T. Chen, J. Botimer, T. Chou, and Z. Zhang, "A 1.87-mm 2 56.9-gops accelerator for solving partial differential equations," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1709–1718, 2020.

[6] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[7] C. Deng, Y. Sui, S. Liao, X. Qian, and B. Yuan, "Gospa: An energy-efficient high-performance globally optimized sparse convolutional neural network accelerator," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1110–1123.

[8] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *The International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 367–382.

[9] A. Gerami and B. Asgari, "Gust: Graph edge-coloring utilization for accelerating sparse matrix vector multiplication," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[10] N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis, "Energy-efficient hybrid analog/digital approximate computation in continuous time," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 7, pp. 1514–1524, 2016.

[11] X. He, S. Pal, A. Amarnath, S. Feng, D.-H. Park, A. Rovinski, H. Ye, Y. Chen, R. Dreslinski, and T. Mudge, "Sparse-tpu: Adapting systolic arrays for sparse matrices," in *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[12] K. Hedge, H. Asghari, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Flectcher, "Extensor: An accelerator for sparse tensor algebra," in *2019 International Symposium on Microarchitecture (MICRO-52)*. IEEE/ACM, 2019, pp. 319–333.

[13] C.-T. Huang, "Ringcnn: Exploiting algebraically-sparse ring tensors for energy-efficient cnn-based computational imaging," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1096–1109.

[14] Y. Huang, N. Guo, M. Seok, Y. Tsividis, K. Mandli, and S. Sethumadhavan, "Hybrid analog-digital solution of nonlinear partial differential equations," in *MICRO*. IEEE, 2017, pp. 665–678.

[15] D. R. Kincaid, T. C. Oppe, and D. M. Young, "Itpackv 2d user's guide," Texas Univ., Austin, TX (USA). Center for Numerical Analysis, Tech. Rep., 1989.

[16] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 821–834.

[17] J. Kung, Y. Long, D. Kim, and S. Mukhopadhyay, "A programmable hardware accelerator for simulating dynamical systems," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 403–415, 2017.

[18] J. H. Lee, B. Park, J. Kong, and A. Munir, "Row-wise product-based sparse matrix multiplication hardware accelerator with optimal load balancing," *IEEE Access*, vol. 10, pp. 64 547–64 559, 2022.

[19] G. Li, W. Xu, Z. Song, N. Jing, J. Cheng, and X. Liang, "Ristretto: An atomized processing architecture for sparsity-condensed stream flow in cnn," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1434–1450.

[20] S. Li, E. Hanson, X. Qian, H. H. Li, and Y. Chen, "Escalate: Boosting the efficiency of sparse cnn accelerator with kernel decomposition," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 992–1004.

[21] H. Lu, L. Chang, C. Li, Z. Zhu, S. Lu, Y. Liu, and M. Zhang, "Distilling bit-level sparsity parallelism for general purpose deep learning acceleration," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 963–976.

[22] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.

[23] J. Mu and B. Kim, "29.2 a 21× 21 dynamic-precision bit-serial computing graph accelerator for solving partial differential equations using finite difference method," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 406–408.

[24] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishana, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE Internation Sumposium on High Performance Computer Architecture*. IEEE, 2020, pp. 58–70.

[25] D. Ramchandani, B. Asgari, and H. Kim, "Spica: Exploring fpga optimizations to enable an efficient spmv implementation for computations at edge," in *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE, 2023, pp. 36–42.

[26] Y. Saad, *Iterative methods for sparse linear systems*. siam, 2003, vol. 82.

[27] SciPy, "List-of-list sparse matrix," 2020, [Online; accessed April-2020].

[28] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gómez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "Nero: A near high-bandwidth memory stencil accelerator for weather prediction modeling," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 9–17.

[29] N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang, "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 766–780.

[30] N. Srivastava, H. Jin, S. Smith, H. Rong, D. Albonesi, and Z. Zhang, "Tensaurus: A versatile accelerator for mixed sparse-dense tensor computations," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 689–702.

[31] J. Tong, A. Itagi, P. Chatarasi, and T. Krishna, "Feather: A reconfigurable accelerator with data reordering support for low-cost on-chip dataflow switching," 2024. [Online]. Available: https://arxiv.org/abs/2405.13170

[32] C. K. Vadlamudi and B. Asgari, "Electra: Eliminating the ineffectual computations on bitmap compressed matrices," *IEEE Computer Architecture Letters*, 2024.

[33] R. W. Vuduc and H.-J. Moon, "Fast sparse matrix-vector multiplication by exploiting variable block structure," in *International Conference on High Performance Computing and Communications*. Springer, 2005, pp. 807–816.

[34] Y. Wang, C. Zhang, Z. Xie, C. Guo, Y. Liu, and J. Leng, "Dual-side sparse tensor core," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1083–1095.

[35] A. Yazdanbakhsh, A. Moradifirouzabadi, Z. Li, and M. Kang, "Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 744–762.

[36] S. Zeng, Y. Lin, S. Liang, J. Kang, D. Xie, Y. Shan, S. Han, Y. Wang, and H. Yang, "A fine-grained sparse accelerator for multi-precision dnn," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 185–185.

[37] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 261–274.

InsightAlign: A Transferable Physical Design Recipe Recommender Based on Design Insights

Hao-Hsiang Hsiao[1], Sudipto Kundu[2], Wei Zeng[2], Wei-Ting J. Chan[3], Deyuan Guo[2], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, GA; [2]Synopsys Inc., Sunnyvale, CA; [3]Synopsys Inc., Hillsboro, OR
{thsiao, limsk}@gatech.edu; {Sudipto.Kundu, Wei.Zeng, Wei-Ting.Chan, Deyuan.Guo}@synopsys.com

***Abstract*—Physical design tools have complex workflows with many different ways of optimizing power, performance, and area (PPA) out of a large number of options and hyperparameters in different engines and functionalities. Black-box optimization techniques are widely adopted to automate quality-of-result (QoR) exploration. Such exploration often proves impractical in real-world customer environments due to high computational demands, lengthy exploration cycles, and the need for large parallel jobs. To reduce the exploration space for viable compute resource requirements, we propose a novel design methodology to enable transferable learning by incorporating design insights crafted on top of physical design experts' experience and streamlining QoR exploration as a sequence generation task for best recipe selection. We apply language model-inspired alignment techniques to learn the ranking of different recipe sets, enabling our model to generalize beyond known-good manually tuned expert design recipes. Extensive evaluations demonstrate our method's superior QoRs and runtime performance on unseen industrial designs and rigorous benchmarks.**

I. Introduction

Advancements in technology nodes boost the capabilities available to circuit designers but, on the other hand, introduce significant challenges at every stage of the physical design flow, driven by an explosion in design complexity. Commercial physical design tools address these challenges through extensive tunability. At each phase of the design process, these tools are meticulously calibrated to resolve specific design- or technology-related issues by balancing critical factors—such as congestion, cell density, and routing constraints—to optimize the quality of result (QoR) at flow signoff. However, as product requirements diversify, these tools are increasingly required to support a wider range of QoR intentions. This growing spectrum of tunable parameters can quickly overwhelm designers, making rapid and optimized design turnarounds ever more challenging.

Although recent works on design flow recipe recommendation have demonstrated progress in using modern optimization techniques to manage the rapidly-growing tool parameters, the search for systematic approaches to align automatic recipe recommenders and design experts' experiences remains open. To address this challenge, we developed a novel automatic flow recipe recommendation framework that comprehends physical design expert knowledge by encoding critical design flow health metrics—what we refer to as **"design insights"**—that go beyond traditional QoR measurements. These insights are the nuanced analyses that experienced designers perform while evaluating and debugging designs, encompassing both high-level flow analysis and detailed engine-level behavior to provide a holistic view of the design's flow trajectory. These expert analyses are systematically transformed into quantitative numerical labels, automatically generated during each place and route (P&R) run, which guide our framework into design-specific recommendations.

Building on insights that capture critical design behaviors, we frame the flow recipe recommendation problem as a sequence generation task. This approach enables us to leverage state-of-the-art large language model (LLM) alignment techniques—traditionally employed to align model outputs with human preferences—to align our model's recipe recommendations with QoR intentions in physical design flows: by incorporating design-specific insights and applying pairwise contrastive training on offline datasets, our model learns to rank recipe sets by their effectiveness, rather than simply memorizing high-performing configurations. This approach provides the model with a nuanced understanding of recipe effectiveness, empowering it to suggest unexplored recipes that align with QoR requirements tailored to each unique design. Additionally, this approach addresses the challenge of exact QoR prediction, which is particularly infeasible for industrial-scale designs due to the variability in design scales and QoR metrics. By focusing on relative effectiveness rather than absolute values, our model achieves robust, cross-design generalization in recipe recommendations. For the first time, LLM alignment techniques are applied to physical design flow recipe recommendation, and extensive experiments demonstrate optimal zero-shot recipe recommendation results, even for previously unseen designs. Our main contributions are as follows:

- We formalize the process of encoding expert knowledge-based health analyses of designs as quantitative labels, creating a comprehensive representation of design flow health metrics that extends beyond traditional QoR measurements. These labels serve as a foundation for informed recipe recommendations across diverse design traits and technology nodes.
- We propose a novel application of LLM alignment techniques to flow recipe recommendation, achieving alignment between recipe recommendations and QoR intentions of the physical design flow. Our approach leverages ranking-based learning to prioritize recipe sets based on relative effectiveness, thereby addressing the limitations of conventional supervised learning approaches that depend on memorizing previously observed recipes. Additionally, this method circumvents the challenge of exact QoR prediction, facilitating adaptable cross-design interoperability.
- This work embodies a tight integration of a flow recommender at an industrial scale, real-world designs and advanced technology nodes on top of an industrial physical design tool [1]. Extensive experimental evaluations demonstrate the model's superior zero-shot performance across unseen designs, highlighting its generalizability and effectiveness in practical deployment environments.

II. Background

Automating the tuning of Electronic Design Automation (EDA) flows has become essential for achieving optimal power, performance, and area (PPA) outcomes as design complexity increases. Due to the vast parameter space of physical design flows in advanced technology nodes, conventional design approach to sweep a limited set of key flow parameters is runtime infeasible and often leaves beneficial parameter combinations unexplored. Recent EDA research has thus focused on leveraging machine learning inspired optimization techniques, each with unique strengths and limitations, to automate this

process. This section summarizes several prominent approaches for automatic flow tuning.

- Bayesian Optimization (BO): BO [2]–[5] uses a Gaussian process surrogate model to predict PPA based on parameter settings, selecting configurations iteratively through an acquisition function. This method excels in high-cost evaluation settings but can be limited by the accuracy of its surrogate model, particularly in highly dynamic or noisy parameter spaces.
- Ant Colony Optimization (ACO): ACO [6] models parameter tuning as a graph search, with parameters represented as nodes and pheromone trails guiding configuration selection. While effective in escaping local optima, ACO may require extensive tuning of pheromone parameters and often converges slowly, making it less suitable for large-scale, high-dimensional searches.
- Recommendation-Based Optimization: Inspired by matrix factorization, this approach [7] assigns latent features to netlists and configurations, predicting QoR through feature similarity. While useful for suggesting high-quality configurations, this method can struggle in high-dimensional spaces and lacks domain-specific insights, potentially leading to suboptimal recommendations.
- Reinforcement Learning (RL)-Based Optimization: Reinforcement learning frameworks, such as that proposed by [8]–[10], iteratively refine expert-provided configurations to improve cumulative PPA rewards. Although effective for fine-tuning, this approach relies heavily on the initial configuration and is limited by episode length, making it less effective for discovering new configurations.

Existing use models of recipe recommender include (1) offline model trained on in-house design project archive, which may not generalize in new designs or technology nodes, and (2) online model that starts with an unexplored design and iteratively collects and trains itself as training data are accumulated through multiple iterations of a design flow, which could consume huge compute resources. Several previous works identified the respective challenges of generality and resource hungriness and suggested building transferable recipe recommender from offline design project archive to enable the kick-start of online learning [3], [4], [6], [9], [11]. As pointed out by [8] and [12], the observability of physical design flow health is crucial to allow recipe recommender to discover design similarity and achieve transferability. However, previous works' flow health metrics are limited to intuitive statistics of design characteristics and categorized QoR metrics. [8] suggests metrics dedicated to only cell placement. [12] and [13] suggest using static snapshots of QoR or resource metrics at each flow stage, which do not capture the natural metric fluctuations at different flow stages. [9] proposes building a graph neural network (GNN) to embed design characteristics for a transferable recommender. However, it does not address the complexity of tool behaviors under different QoR objectives.

III. Methodologies

The goal of our flow recipe recommendation framework, as depicted in Figure 1, is to select the optimal subset of recipes to meet multiple user-defined design objectives. Given a set of preconfigured recipes, the task is to identify the best subset of recipes that will yield superior QoR according to user intentions.

A. Problem Formulation

InsightAlign contains the following key components:

a) Design insights: Contextual insights from the prior run learning contain fine-grained real-time analysis of the complex workflow of design implementation flow. Typically, physical design tool users observe the final metrics in terms of timing (worst negative slack (WNS) and total negative slack (TNS)), area, power, routing design rule check (DRC) violations, etc. Conventional black-box optimization around the tool explores solution space based on the final metrics in an iterative loop. In our recommendation framework formulation, one of the key components is the ability to consume deep insights into the design, as well as the flow that is learned from the first run of the design. Typical examples of such insights are shown in Table I. The tool natively discovers timing, area, and power challenges in the form of insights. Our objective is to drive the recommendation framework with deep insights as an input, so that it overcomes the limitation of black-box optimization. Each time the P&R flow iteration is executed, the deep insights are collected and fed to the recommendation framework to suggest more relevant recipes that will provide faster convergence to the best PPA.

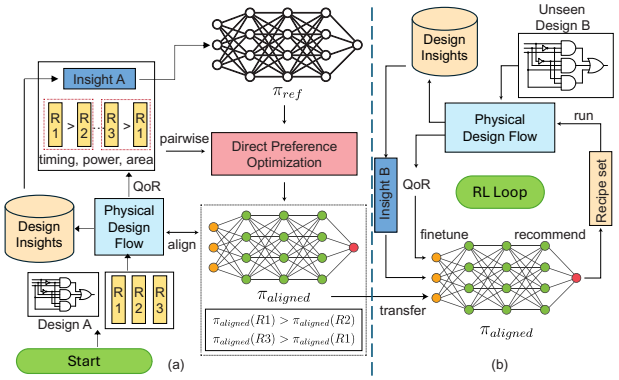


Fig. 1. Overview of InsightAlign: (a) offline alignment, (b) online fine-tuning.

TABLE I
Examples of Insights

Category	Insight Description	Range
Placement	Congestion level during placement step X	{low, medium, high}
Timing	Is easy to meet timing constraints	{yes, no}
Power	Good opportunity for power saving during step Y	{yes, no}
Power	Sequential-cell power is dominant	{yes, no}
Power	Leakage power is dominant	{yes, no}
Clock	Critical paths with harmful clock skew	{yes, no}
Timing	Instance count from hold-time fixes	$\mathbb{N}$
Timing	Weak cell percentage on critical paths	$\mathbb{R} \in [0.0, 100.0]$

TABLE II
Examples of Recipes

Category	Recipe Description
Design intention tradeoffs	Adjust tradeoffs among timing, power, and area metrics
Timing	Balance weights of early hold- and setup-time fixing, and placement perturbations
Clock tree	Adjust clock-tree synthesis (CTS) hyperparameters for tradeoffs among timing, skew and latency
Routing	Adjust knobs of routing congestion
Routing	Adjust global routing hyperparameters

b) Recipe: Commercial physical design tools usually offer thousands of flow parameters and options, creating an expansive search space that is infeasible to explore exhaustively. To manage this, we prune the space by defining a suite of preconfigured options, referred to as "recipes." Consisting of a set of flow parameters and options, each recipe serves a specific QoR intention. Example recipes are outlined in Table II. While each individual recipe has dedicated intention, we aim to capture the complex interactions among these

TABLE III
DETAILS OF INSIGHTALIGN MODEL ARCHITECTURE AND DIMENSIONS

Layer	Type	Input Size	Output Size
Decision Token Embed.	Embedding	$(40, 3)$	$(40, 32)$
Recipe Pos. Enc.	Positional Encoding	$(40, 32)$	$(40, 32)$
Insight Embed.	Linear $\times 1$	$(1, 72)$	$(1, 32)$
Transformer Dec.	Transformer Decoder $\times 1$	$(1, 32)$ $(40, 32)$	$(40, 1)$
Probabilistic	Sigmoid $\times 40$	$(40, 1)$	$(40, 1)$

recipes, and to combine recipes strategically to achieve multiple design objectives simultaneously.

B. Stage-Based Flow Recipe Recommender

From initial cell placement to design closure, each stage in physical design flow deals with different physical design problems and usually provides diverse parameters. Our framework provides separate recipe sets for each flow stage to fully leverage the parameter diversities, as shown in Figure 2. Note that our staged framework is not specific to a given stage definition and can be generalized to any flow variation. We refer to each run of physical design flow loaded with proposed recipe sets as an **iteration**. Each iteration can include up to N recipe sets, determined by the available computational resources.

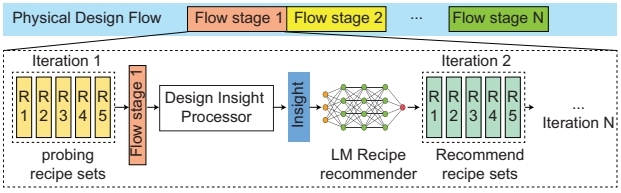


Fig. 2. InsightAlign incorporated into the physical design flow. Each "Rx" represents a recipe set, which consists of a subset of preconfigured recipes.

Our recommender intelligently consumes the first iteration, serving as an insight-probing phase, to generate initial insights. We refer to this insight-probing phase as "**offline alignment**." During offline alignment, our design insight analyzers, imitating human experts' flow probing process, perform analysis in various algorithm engines, monitor flow trajectory and accumulate insights in a non-volatile storage. After the first iteration, subsequent iterations leverage these accumulated insights and prior QoR results to iteratively refine the recipe sets, which we refer to "**online fine-tuning**." With each new iteration, additional insights are gathered, providing a progressively generalized view of the design. As more recipe sets are explored through successive iterations, the recommender becomes increasingly tailored, resulting in optimized QoR outcomes.

C. Language Model-Based Recipe Recommendation

Our recipe recommender is based on a decoder-only generative language model architecture, where recipes are treated as tokens and predicted sequentially in an autoregressive manner, as shown in Figure 3. The recipes are evaluated one by one at each time step t, and the model estimates the probability of selecting each recipe in the recommended subset based on the preceding context. This autoregressive process enables the model to construct recipe subsets progressively, adapting its choices to the design insights and previously chosen recipes. Our model includes the following components with details in Table III:

1) Decision Token Embedding: Each recipe decision r_t indicates whether a recipe is selected or not. We train separate embeddings for both decision, allowing the model to learn nuanced representations of each decision outcome in the context of recipe selection, i.e., "selected" (1), "not selected" (0), and an additional "SOS" (start-of-sequence) token for initiating the generation process.
2) Recipe Positional Embedding: Positional encoding is essential in language models to convey the order of tokens, and here it is used to represent the order of recipes in the tuning sequence. The decision token embedding is combined with a recipe-specific positional encoding, which provides necessary information such that the model can distinguish among recipes based on both their unique characteristics and their positions in the sequence.
3) Insight Embedding: Design insights are concatenated into a vector and processed by a linear layer to produce a high-dimensional embedding that will be further served as a contextual information for design specific recipe recommendation.
4) Transformer Decoder Layer: The model uses a single-head Transformer decoder layer [14] that leverages contextual information through cross attention between recipe and design insight embeddings. This architecture enables the model to learn interdependencies between recipe decisions to adaptively tune recipes based on previous decisions and design insights.
5) Probabilistic Layer: Finally, after the decoder layer generates a contextualized representation of each recipe, the model converts this representation into a probability for each recipe using a sigmoid activation function. This probability indicates the likelihood of selecting each recipe, enabling the model to make probabilistic, informed recommendations based on the embeddings and contextual analysis performed in prior layers.

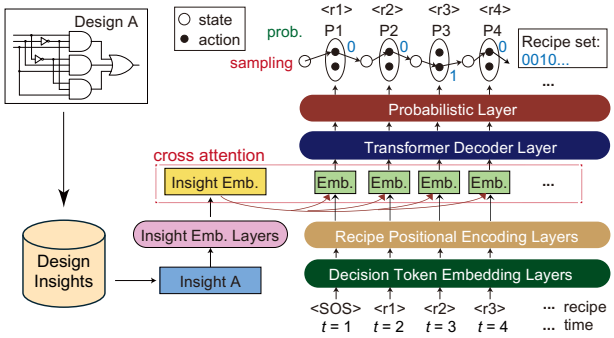


Fig. 3. InsightAlign model architecture for recipe set recommendation. The same model is used in offline alignment and online fine-tuning phases.

D. Model Training Overview

Ideally, tuning would be fully "online," using the QoR of each recipe set to directly guide model updates and the selection of subsequent recipes. However, the extended runtime of the commercial physical design tool—which can take days to weeks for industrial-scale designs—makes continuous online updates impractical. To address this, we adopt a two-stage training strategy.

Our model training involves two phases: (1) Offline alignment and (2) Online fine-tuning. The offline alignment process is detailed in Algorithm 1. Our framework performs offline preference-based alignment by training a language model to assign higher probabilities to recipes that historically achieved better QoR outcomes. For each design, all known recipe sets are pairwise compared based on their QoR values, and the model is optimized using margin-based direct preference optimization (DPO) [15] to reflect these preferences by assigning higher probabilities to recipe sets with better QoRs. During inference, the aligned model employs beam search with width K to generate the top-K recommended recipe sets for an unseen design

by keeping track of the K most promising partial sequences at each step. After offline alignment, the model enters the online fine-tuning phase, where it is further refined for specific designs. Here, the model continuously evolves based on direct feedback from the physical design tool, using immediate QoR evaluations of newly recommended recipe sets to make real-time adjustments. This online learning phase allows the model to adapt dynamically, providing fine-tuned recipe recommendations tailored to each design's unique characteristics.

Algorithm 1 Offline QoR-Alignment

Input:
Dataset $\mathcal{D} = \{(I^i, R^i, Q^i)\}_{i=1}^N$, where design insight $I^i \in \mathcal{I}$, recipe set $R^i \in \{0,1\}^n$, compound QoR score $Q^i \in \mathbb{R}$
Beam width K, Insight of a new design $I' \in \mathcal{I}$
Output: Policy π_ϕ and top-K recipe recommendations $\{\hat{R}_k\}_{k=1}^K$
function AlignmentTrain($\mathcal{D}$)
  Initialize policy parameters ϕ
  while *not converged* **do**
    for $I^k \in \mathcal{D}$ **do**
      $\mathcal{P}_k = \{(R^i, R^j) : I^i = I^j = I^k\}$ ▷ All recipe pairs for design k
      for $(R^i, R^j) \in \mathcal{P}_k$ **do**
        $(R_w, R_l) \leftarrow (R^i, R^j)$ if $Q^i > Q^j$ else (R^j, R^i)
        $\log \pi_\phi(R \mid I^k) = \sum_{t=1}^n \log P(r_t \mid r_{<t}, I^k; \phi)$ for $R \in \{R_w, R_l\}$
        Evaluate $\mathcal{L}_{\text{MDPO}}(\phi)$ with (2)
        $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_{\text{MDPO}}(\phi)$ ▷ Gradient descend
      end for
    end for
  end while
  return π_ϕ
end function
function BeamSearch(π_ϕ, I', K)
  $\mathcal{B}_0 \leftarrow \{([\text{SOS}], 0)\}$ ▷ Initial beam: (sequence, log probability)
  for $t = 1$ to n **do**
    $\mathcal{B}_t \leftarrow \emptyset$
    for $(R_{<t}, s) \in \mathcal{B}_{t-1}$ **do**
      $P_t \leftarrow \pi_\phi(\cdot \mid R_{<t}, I')$
      for $r_t \in \{0,1\}$ **do**
        $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \{(R_{<t} \oplus r_t, s + \log P_t(r_t))\}$
      end for
    end for
    $\mathcal{B}_t \leftarrow$ top-K sequences from $\mathcal{B}_t$ by score
  end for
  return $\{R : (R, s) \in \mathcal{B}_n\}$ ▷ Return top-K sequences
end function
$\pi_\phi \leftarrow$ AlignmentTrain($\mathcal{D}$)
$\{\hat{R}_k\}_{k=1}^K \leftarrow$ BeamSearch(π_ϕ, I', K)
return $\pi_\phi, \{\hat{R}_k\}_{k=1}^K$

E. Offline Alignment

Our recommendation framework leverages alignment techniques inspired by large language model (LLM) [14] to predict optimal recipe sets that enhance QoR for complex industrial designs. Instead of relying on direct QoR prediction—which is challenging due to the variability in design scales and inherent complexity—our approach employs a preference-based strategy. This allows the model to make effective recipe recommendations by comparing relative performance, even when absolute QoR values are difficult to estimate. This reframing allows the model to focus on the relative performance among recipe sets, rather than attempting to predict absolute QoR values. The overall process is illustrated in Figure 1(a).

1) Offline Data Collection and Preference-Based Training: We first constructed an offline dataset using the commericial P&R tool. The dataset contains multiple (design, recipe set, QoR) combinations. Each data point is composed of (1) a design in the form of the insight vector with numerical and categorical data describing the contextual design knowledge, (2) a set of recipes selected for this run, and (3) the QoR outcome after applying that recipe set.

To align the model's recipe predictions with actual QoR-based preferences, we perform pairwise probability updates strategy. The process starts by pairing the data points from the same design. That is, for each design (insight), we compare pairs of recipe sets with known QoR outcomes, querying the model the probability of generating each recipe set, and then train the model to favor (modify model weights to assign higher probability to) recipes that contribute to better performance. This alignment process is iterated through all the pairs and all the designs. Each pairwise comparison guides the model to prioritize configurations that historically produce higher-quality results, ultimately tuning the model to reflect the QoR preferences.

2) Marginal Direct Preference Optimization: Traditional LLM alignment through Reinforcement Learning from Human Feedback (RLHF) [16] employs a two-stage approach: first training a reward model to evaluate outputs based on human-defined quality metrics, then utilizing proximal policy optimization (PPO) [17] to iteratively optimize a policy network. In contrast, DPO streamlines this process into a single-stage training paradigm by introducing an optimized loss function, eliminating the need for a separate reward model. The DPO loss $\mathcal{L}_{\text{DPO}}(\phi)$ is given by

$$-\mathbb{E}_{(R_w, R_l, I)\sim\mathcal{D}}\Big[\log\sigma\Big(\beta\log\frac{\pi_\phi(R_w \mid I)}{\pi_{\text{ref}}(R_w \mid I)} - \beta\log\frac{\pi_\phi(R_l \mid I)}{\pi_{\text{ref}}(R_l \mid I)}\Big)\Big] \tag{1}$$

Here, R_w and R_l denote the preferred ("winning") and non-preferred ("losing") recipe set in pairwise comparisons under design insight I. The probability of outputting R given design insight I is represented by $\pi_\phi(R \mid I)$ under model parameters ϕ, while $\pi_{\text{ref}}(R \mid I)$ denotes the reference policy probability. The preference model's sharpness is controlled by hyperparameter $\beta > 0$, and $\sigma(\cdot)$ represents the sigmoid function. While standard DPO effectively learns recipe rankings, it does not account for preference magnitude differences. A margin-based modification [18] addresses this by assuming policy preferences scale exponentially with recipe QoR scores and adopting a uniform reference policy $\pi_{\text{ref}}(R \mid I)$ to balance exploration and exploitation. The margin-based DPO loss is given by

$$\mathcal{L}_{\text{MDPO}}(\phi) = \mathbb{E}_{(R_i, R_j, I)\sim\mathcal{D}}\Bigg[\max\Big(0, \lambda(\text{QoR}(R_i) - \text{QoR}(R_j)) - \text{sign}(\text{QoR}(R_i) - \text{QoR}(R_j)) \cdot (\log\pi_\phi(R_i \mid I) - \log\pi_\phi(R_j \mid I))\Big)\Bigg] \tag{2}$$

3) Evaluate Probabilities of Recipe Selections: To compute the margin-based DPO loss in (2), we should be able to efficiently compute $\pi_\phi(R_i)$, the likelihood that each recipe set R_i will be predicted by our recipe model. The decoder-only language model is particularly well-suited for this task because it allows us to efficiently compute the likelihood of any recipe set by leveraging teacher forcing [19] in the training process. This setup is illustrated in Figure 4.

In the teacher forcing process, we input each recipe step by step as ground truth to guide the model's next prediction. Specifically, a recipe set, represented as a sequence $R = [r_1, r_2, \ldots, r_n]$ (with "1" indicating a recipe is selected and "0" indicating it is not), serves as

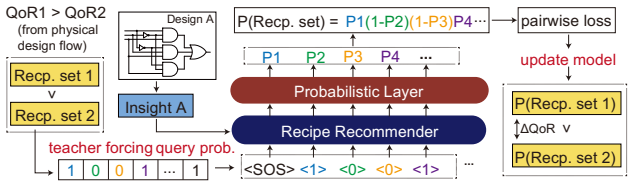


Fig. 4. The process of teacher forcing used in offline alignment.

input. At each time step $t = 1, \ldots, n$, the model receives the actual recipe r_t, which enables it to compute the conditional probability:

$$\log \pi_\phi(R \mid I^k) = \sum_{t=1}^{n} \log \pi_\phi(r_t \mid r_{<t}, I^k; \phi) \tag{3}$$

In this manner, the model can efficiently evaluate the probability of each recipe set and hence the pairwise margin-based DPO loss (2).

F. Recipe Set Recommendation

To recommend recipe sets, we employ beam search [19] with width K to identify the top-K recipe sets following our trained policy distribution. At each time step t, given an unseen design insight I' in the insight space $\mathcal{I}$, the algorithm maintains K partial sequences and extends each by sampling according to the policy probabilities $\pi_\phi(r_t \mid r_{<t}, I'; \phi)$, shown as "sampling" in Figure 3. The search accumulates log probabilities of selected recipes, retaining the K sequences with highest cumulative scores. This process continues until all recipe decisions are made, yielding K complete recipe sets that best align with our QoR-optimized policy distribution.

G. Online Fine-Tuning

After the offline alignment phase, the same model transitions into an online fine-tuning stage to further optimize its recipe recommendations for specific designs, as illustrated in Figure 1(b). Based on the offline learned policy, the model continuously adapts its policy π_ϕ through real-time feedback from the physical design tool. Each iteration follows a closed-loop learning system: the model proposes recipe sets, executes the physical design flow, and updates its parameters based on observed QoR metrics. This adaptive mechanism enables the model to capture design-specific optimization patterns that may not have been present in the offline training data, enhancing its ability to generate highly specialized recipes that better align with each design's unique characteristics. We implement batch online learning where the model proposes $K = 5$ recipes in each iteration and leverages both margin-based DPO [18] and PPO loss [17] to fine-tune the model.

IV. Experimental Results

A. Experiment Setup

We implemented our proposed recipe recommendation framework using *Python* and the deep learning library *PyTorch* [20]. To validate the effectiveness of our framework, we conducted experiments across 17 industrial-scale, real-world benchmarks. These benchmarks span a diverse range of design categories and advanced technology nodes, from 45 nm to sub-10 nm processes with gate counts up to 2 million, to ensure comprehensive evaluation under realistic conditions. Our optimization framework integrates $n = 40$ distinct recipes, each calibrated for distinct design objectives. The primary goal of the framework is to identify an optimal subset of recipes that collectively maximize a user-defined QoR intention as a compound score s based on multiple performance metrics m_i weighted by w_i, i.e.,

$$s = \sum_i w_i g_i (m_i - \text{mean}(m)_i)/\text{std}(m)_i \tag{4}$$

where $\text{mean}(m)_i$ and $\text{std}(m)_i$ are the mean and the standard deviation, respectively, of the ith metric over all datapoints of the same design, $g_i = 1$ if metric i is to be maximized, and -1 otherwise.

For illustration purposes, in this paper we define the QoR intention as minimizing the total power and TNS, with weights 0.7 and 0.3, respectively. In Algorithm 1, we use hyperparameter $\lambda = 2$ and beam width $K = 5$ in offline alignment and online fine-tuning.

We design different experiments to evaluate the efficacy of each phase. For the offline alignment phase, we construct a dataset $\mathcal{D}$ using a commercial physical design tool [1], comprising 3,000 datapoints collected from 17 designs with various recipe combinations. For the online fine-tuning phase, we iterate the recommendation and fine-tuning process for two designs on top of different offline alignment outcomes, and show how their QoRs are further improved and converged throughout iterations.

B. Offline Alignment Result

One of the key objectives of our work is to enable the framework to generalize optimization strategies to previously unseen designs without requiring retraining. To evaluate the model's transferability, we conducted zero-shot evaluations—meaning we tested the model on entirely new designs without any additional training or fine-tuning—on designs excluded from the training phase. This evaluation tests the model's ability to apply learned knowledge to new designs.

To rigorously evaluate the effectiveness of the offline phase, we employed a k-fold cross-validation strategy [21]. We divide the 17 designs into $k = 4$ random groups with roughly equal numbers of datapoints and carry out k iterations of evaluation. In iteration i, all designs in the ith group are held out for testing, and only the remaining designs are used for training and tuning the model. (As a result, we use different trained models to test designs in different groups to ensure all designs are unseen when evaluated.) By systematically rotating the combinations of training and testing splits, this approach ensures that all designs are evaluated once as unseen by the corresponding model. Therefore, the performance of the framework is assessed across the entire design spectrum.

Table IV summarizes the results of the zero-shot evaluation with cross-validation. For each design, we take five distinct recommendations from the model and compare the QoR in terms of power, TNS, and the compound score against the best-known datapoints of the same design in our offline dataset. (Recall that with cross-validation, the model never sees any datapoint from the design being tested.) The "Win%" column indicates the percentage of known recipe sets (out of approximately 200) that are outperformed by the best of five recommended recipe sets. It shows that the framework demonstrated performance comparable to, or even exceeding, the best-known recipe sets, without any fine-tuning.

Figure 5 visualizes the power-timing distribution of our zero-shot recommendations for four unseen designs: D4, D6, D11, and D14, compared to the QoR distribution of known recipe sets (unseen by the model). The recommended recipe sets are concentrated in the lower-left region of the plot, indicating superior performance with reduced power consumption and negative slacks. In contrast, the known recipe sets from the offline dataset exhibit a scattered distribution, reflecting the significant cost and effort required for extensive exploration, as opposed to our approach, which achieves optimal or near-optimal results at the very beginning of the design iteration.

These observations underscore the effectiveness of the design insight representation in capturing transferable knowledge, enabling the direct application of learned optimization strategies across designs. Additionally, they confirm the ability of our framework to capture the

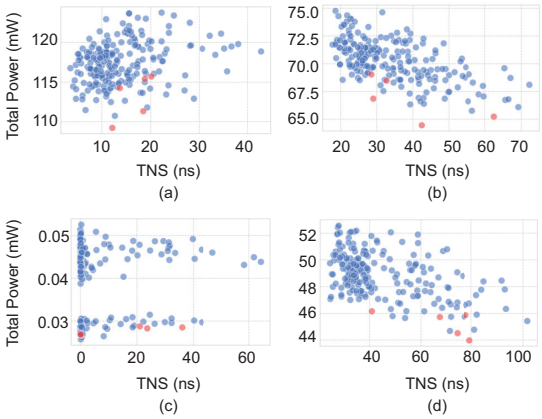


Fig. 5. QoRs of zero-shot recommended (red) compared with all known recipe sets in the dataset (blue) for designs (a) D4, (b) D6, (c) D11, (d) D14.

TABLE IV
EVALUATION OF OFFLINE ALIGNMENT ON UNSEEN DESIGNS BASED ON CROSS VALIDATION. "QOR SCORE" IS OUR OPTIMIZATION OBJECTIVE DEFINED BY (4). "WIN%" INDICATES THE PERCENTAGE OF KNOWN RECIPE SETS THAT OUR BEST RECOMMENDATION SURPASSES.

Design	Best known recipe set			Offline alignment			
	TNS (ns)	Power (mW)	QoR Score	TNS (ns)	Power (mW)	QoR Score	Win%
D1	20.23	1513.39	1.94	16.71	1539.99	1.42	98.7
D2	25.14	1177.70	2.12	55.20	1192.72	1.42	96.9
D3	18.28	2054.68	1.60	27.73	2046.93	1.70	100.0
D4	6.62	111.48	1.96	12.11	109.23	2.34	100.0
D5	0.0552	192.12	1.38	0.0823	191.56	1.49	100.0
D6	39.24	66.34	1.45	42.45	64.46	2.11	100.0
D7	18.12	143.17	1.40	14.47	144.00	1.44	100.0
D8	3.88	75.78	1.28	1.43	75.29	1.90	100.0
D9	0.816	640.01	1.98	0.824	614.90	2.65	100.0
D10	72.24	12.21	1.75	235.98	12.71	0.74	88.5
D11	0.157	0.0257	1.50	0.115	0.0268	1.40	98.1
D12	0.230	403.79	1.53	0.931	399.71	1.61	100.0
D13	118.70	311.99	3.54	379.85	338.46	0.56	95.2
D14	71.24	44.68	1.41	79.39	43.97	1.57	100.0
D15	0.0354	642.84	1.59	0.0838	640.65	1.68	100.0
D16	0.0000	0.706	1.66	0.0012	0.708	1.56	98.6
D17	282.06	688.48	1.11	793.51	677.32	0.93	97.8

complex interaction among recipes, recommend undiscovered, good-quality combinations of recipes, and provide excellent starting points of design iterations for previously unseen designs.

C. Online Fine-Tuning Result

Online fine-tuning provides an additional option to further tailor recipes for specific designs, enabling the achievement of optimal QoRs. Building on top of the offline alignment model, we show the online fine-tuning result on designs D10 and D6. As shown in Table IV, D10 is a design with slightly worse zero-shot recommendation compared to other benchmarks. Design D6, on the contrary, starts from a superior recommendation at the very beginning.

For design D10, our online reinforcement learning progressively refines the strategy for this specific design, allowing the model to recommend recipes that yield increasingly higher QoR scores over iterations. Figure 6(a) illustrates the average QoR score of the top five recipes encountered so far across iterations, highlighting the model's ability to continuously discover better-performing recipes. Additionally, Figure 7 visualizes the progressive QoR distribution over online learning. Notably, by leveraging the offline alignment knowledge, our model begins with a strong initial point. Over successive iterations,

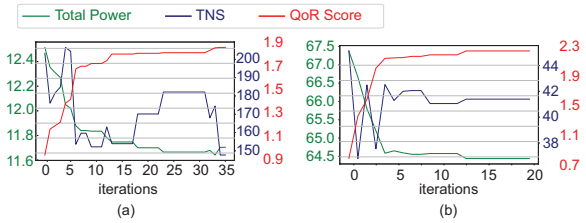


Fig. 6. Online fine-tuning trajectory: total power (lower-better), TNS (lower-better), and QoR score (higher-better) per iteration for (a) D10 and (b) D6.

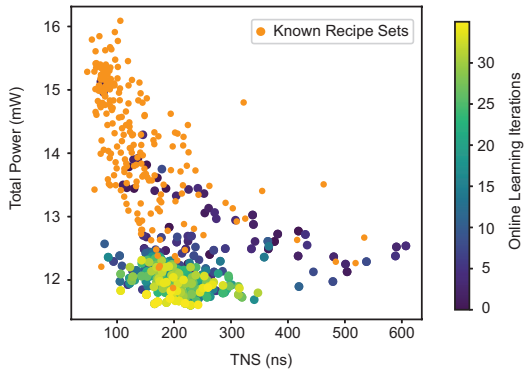


Fig. 7. Scatter plot of QoR for D10 in online fine-tuning phase. Early iteration points (darker colored) scattered on upper-right quickly move to lower-left (lighter colored) and converge to a better QoR than all known recipe sets.

the recommended recipes transition from the upper-right to the lower-left region of the plot and after a small number of iterations, surpassed all best-known recipe sets despite its suboptimal starting points. This demonstrates the continuous enhancement and provides designers with the flexibility to balance further QoR improvements against the cost of additional design iterations. For design D6, as shown in Figure 6(b), the online fine-tuning achieves even better QoR results than the zero-shot recommendation, and converges within even fewer iterations than D10, thanks to its superior starting points from offline alignment. These observations show that a good offline alignment can significantly reduce the number and hours of design iterations. With design-specific insights and user intention built in our framework, the online fine-tuning can achieve better QoR tailored to the specific design and different user intentions on top of the offline stage.

V. CONCLUSION

This work introduces a novel flow recipe recommendation approach that enables transferable learning by combining expert design insights with LLM alignment technique. Once the offline alignment model is established, online fine-tuning by running the physical design tool can achieve faster PPA closure by connecting recipe relevance with newly generated insights and QoRs. The combination of all these techniques addresses the challenge of high turnaround time and large compute demand, which is the most critical bottleneck of automatic flow tuning using expensive black-box tool evaluations. Extensive experiments on industrial-scale designs and advanced technology nodes demonstrate superior zero-shot PPA performance and fast-converging design iterations across unseen designs.

VI. ACKNOWLEDGEMENTS

We would like to express our gratitude to Vishal Khandelwal and Ravishankar Rao from Synopsys, Inc. for their support and valuable discussions.

REFERENCES

[1] Synopsys, Inc., "Fusion Compiler." [Online]. Available: https://www.synopsys.com/implementation-and-signoff/physical-implementation/fusion-compiler.html

[2] Y. Ma, Z. Yu, and B. Yu, "CAD tool design space exploration via Bayesian optimization," in *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, 2019, pp. 1–6.

[3] H. Geng, Q. Xu, T.-Y. Ho, and B. Yu, "PPATuner: Pareto-driven tool parameter auto-tuning in physical design via Gaussian process transfer learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1237–1242.

[4] Z. Zhang, T. Chen, J. Huang, and M. Zhang, "A fast parameter tuning framework via transfer learning and multi-objective Bayesian optimization," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 133–138.

[5] H. Geng, T. Chen, Y. Ma, B. Zhu, and B. Yu, "PTPT: Physical design tool parameter tuning via multi-objective Bayesian optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 178–189, 2023.

[6] R. Liang, J. Jung, H. Xiang, L. Reddy, A. Lvov, J. Hu *et al.*, "FlowTuner: A multi-stage EDA flow tuner exploiting parameter knowledge transfer," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[7] M. Kazda, M. Monkowski, and G. Antony, "APEX: Recommending design flow parameters using a variational autoencoder," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 2023, pp. 1–6.

[8] A. Agnesina, K. Chang, and S. K. Lim, "VLSI placement parameter optimization using deep reinforcement learning," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[9] H.-H. Hsiao, Y.-C. Lu, P. Vanna-Iampikul, and S. K. Lim, "FastTuner: Transferable physical design parameter optimization using fast reinforcement learning," in *Proceedings of the 2024 International Symposium on Physical Design*, 2024, pp. 93–101.

[10] D. Luo, Q. Sun, Q. Xu, T. Chen, and H. Geng, "Attention-based EDA tool parameter explorer: From hybrid parameters to multi-QoR metrics," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[11] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany *et al.*, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 19–25.

[12] J. Jung, A. B. Kahng, S. Kim, and R. Varadarajan, "METRICS2.1 and flow tuning in the IEEE CEDA robust design flow and OpenROAD ICCAD special session paper," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[13] E. Ustun, S. Xiang, J. Gui, C. Yu, and Z. Zhang, "LAMDA: Learning-assisted multi-stage autotuning for FPGA design closure," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 74–77.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1–11.

[15] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[16] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei *et al.*, "Fine-tuning language models from human preferences," 2020, *arXiv:1909.08593*.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[18] J. Wu, X. Wang, Z. Yang, J. Wu, J. Gao, B. Ding *et al.*, "α-DPO: Adaptive reward margin is what direct preference optimization needs," 2024, *arXiv:2410.10148*.

[19] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," 2016, *arXiv:1610.09038*.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 1–12.

[21] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

Hardware-Software Co-design for Distributed Quantum Computing

Ji Liu*§, Allen Zang†§, Martin Suchara‡, Tian Zhong†, and Paul D Hovland*
*Argonne National Laboratory †The University of Chicago ‡Microsoft Azure Quantum §Equal contribution
{ji.liu, hovland}@anl.gov, {yzang, tzh}@uchicago.edu, msuchara@microsoft.com

***Abstract*—Distributed quantum computing (DQC) offers a pathway for scaling up quantum computing architectures beyond the confines of a single chip. Entanglement is a crucial resource for implementing non-local operations in DQC, and it is required to allow teleportation of quantum states and gates. Remote entanglement generation in practical systems is probabilistic, has longer duration than that of local operations, and is nondeterministic. Therefore, optimizing the performance of probabilistic remote entanglement generation is critically important for the performance of DQC architectures. In this paper we propose and study a new DQC architecture that combines (1) buffering of successfully generated entanglement, (2) asynchronously attempted entanglement generation, and (3) adaptive scheduling of remote gates based on the entanglement generation pattern. We show that our hardware-software co-design improves both the runtime and the output fidelity under a realistic model of DQC.**

I. Introduction

Quantum computing is able to efficiently solve important classes of computational problems that are hard to solve using classical computers [1]. To make the computational power of quantum computers beneficial for real-world applications, quantum computing platforms need to be scaled beyond the size of state-of-the-art systems with limited number of qubits that have been demonstrated so far. However, placement of a large number of qubits within one monolithic quantum processing unit is expected to result in system performance degradation due to cross-talk between nearby physical qubits, limited capability of storing the physical qubits on a single chip with a small footprint, and increased complexity of controlling the individual qubits. Scaling up monolithic QPUs will be therefore constrained by various technical obstacles that are difficult to overcome.

In contrast to monolithic QPUs, distributed quantum computing (DQC) [2]–[4] is a paradigm which aims to interconnect multiple monolithic QPUs together with the goal to allow evaluation of quantum circuits whose sizes exceed the number of qubits in a single QPU. DQC is viewed as an important milestone in scaling quantum computers, as described in the roadmaps of many quantum hardware manufacturers [5], [6]. DQC allows finding the sweet spot for the size of the individual QPUs, and therefore mitigates the aforementioned monolithic QPU scaling difficulties. At the same time, DQC introduces new challenges. Entanglement between the QPUs is the most important resource in DQC, which is consumed to implement remote multi-qubit gates. The generation of remote entanglement involves photon transmission, photonic Bell state measurement for heralding of success, and feedforward of measurement results. These processes result in a probabilistic success of remote entanglement generation and a longer cycle time for each attempt compared to local quantum operations. These entanglement generation features represent a significant bottleneck in the DQC performance. Implementation of remote gates requires prior successful entanglement generation, leading to potential gate delays that may further cascade through the system due to gate dependencies. When initialized qubits sit idle, this introduces additional decoherence leading to circuit output fidelity degradation as well as slower evaluation of the circuit.

In this paper, we study a new DQC architecture which incorporates hardware-software co-design. The three key principles of our architecture are:

1) Use of three different types of hardware qubits: communication qubits which are used for remote entanglement generation, buffer qubits which store the successfully generated entangled states, and data qubits which are used for quantum circuit evaluations.
2) Asynchronous attempts to generate remote entanglement. This smoothens the temporal pattern of successfully generated entanglement, leading to better time resource management.
3) Division of the quantum circuit into segments and identification of equivalent variants of circuit segments obtained by commuting remote gates to allow more efficient adaptive scheduling of remote gates by using information about the success of entanglement generation.

This paper is organized as follows: In Sec. II we review the necessary background and survey related work. We describe our architectural framework in Sec. III, and the evaluation methodology in Sec. IV. Our evaluation results are presented in Sec. V. Finally, we conclude in Sec. VI.

II. Background

In this section, we review basics of quantum information, the mechanism of heralded remote entanglement generation, and the implementation of basic non-local operations in DQC. Then we review relevant literature.

A. Quantum information basics

Quantum information is encoded in continuous complex probability amplitudes. Noiseless (pure) quantum states are linear combinations of eigenstates of the physical systems, i.e. quantum superpositions. Quantum states can be written as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$, where $|0\rangle$ and $|1\rangle$ are orthonormal bases of a two-dimensional Hilbert space, and normalization requires $|\alpha|^2 + |\beta|^2 = 1$. In quantum computing, an entire quantum circuit is a unitary operator U, s.t. $U^\dagger U = UU^\dagger = I$, where the dagger in the superscript denotes composition of complex conjugation and transpose, and I is the identity operator.

Entanglement is another key feature of quantum mechanics, and also one of the most important resources in DQC. A canonical example of entangled states are the 4 orthonormal *Bell states (EPR pairs)* of two qubits: $|\Phi^\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle)$, $|\Psi^\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$. The Bell states can be transformed into each other via applications of single-qubit operations known as *Pauli operators* which are defined as follows: $X|0\rangle = |1\rangle$, $X|1\rangle = |0\rangle$ (also denoted as $\oplus$), $Z|0\rangle = |0\rangle$, $Z|1\rangle = -|1\rangle$, and $Y = iXZ$.

Environment perturbations will cause decoherence by imposing errors on quantum states. Decoherence generally transforms quantum states to mixed states, i.e. a statistical mixture of pure quantum

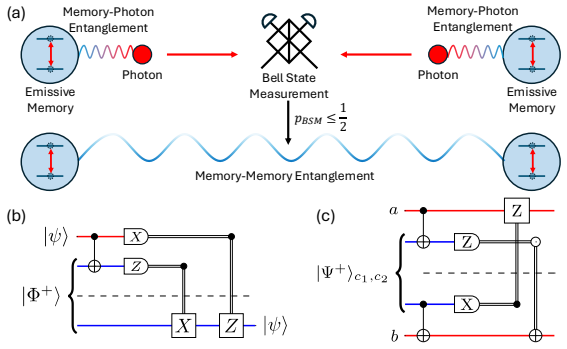


Fig. 1. Preliminaries of DQC. (a) Heralded remote entanglement generation. (b) Quantum state teleportation circuit. (c) CNOT teleportation circuit. The dashed lines denote the partition of two local parties. The blue lines represent the Bell pair resource and the red lines represent data qubits.

states. A common error model is the Pauli channel that applies each of the three Pauli operators and the identity operator with certain probabilities. The *fidelity* of the mixed output quantum state ρ with respect to the ideal pure output state $|\psi\rangle$: $F = \langle\psi|\rho|\psi\rangle$, is a key performance metric.

B. Heralded remote entanglement generation

Heralded remote entanglement generation [7]–[10] can establish entanglement between two local systems that have no direct physical interaction. The effective interaction is realized in the following way:

1) The local systems emit photons which are entangled with themselves. Such matter-photon entanglement can be generally expressed as a standard Bell state.
2) The two emitted photons are transmitted through a physical channel to an intermediate photonic Bell state measurement (BSM) station.
3) The photonic BSM realizes an effective entanglement swapping [11], which conditioned on success projects the quantum state of two remote systems into a Bell state.
4) The classical BSM result is communicated with the two end nodes, which heralds whether the remote entanglement generation is successful.

The above processes are visualized in Fig. 1(a).

C. Nonlocal operations with entanglement

In DQC, nonlocal multi-qubit gates between QPUs can be implemented through teleportation. We may either teleport qubits [12] from one QPU to another QPU, perform local multi-qubit gates and teleport the qubit back, or directly teleport the multi-qubit gates [13]. These approaches require Bell pairs as resources that are consumed for each attempt, local operations on each QPU, and potential feedforward Pauli frame correction. The circuit for quantum state teleportation is depicted in Fig. 1(b). Fig. 1(c) shows the specific circuit that teleports a CNOT gate.

D. Related work

Related to this work, AutoComm [14] and QuComm [15] are state-of-the-art approaches which take into account quantum circuit structure to harness the benefits of DQC. However, these works ignore one important feature - the probabilistic nature of EPR pair generation. AutoComm identifies burst communication patterns in the input program and utilizes state or CNOT teleportation to allow remote operations. It compiles the circuit and schedules remote operations offline, making it unaware of the real-time EPR pair generation pattern. QuComm identifies multi-node collective communication, and uses "buffer" qubits which store generated EPR pairs while communication qubits are freed up to be able to keep generating EPR pairs.

Several research papers have sought to reduce the communication overhead of distributed quantum programs by exploring various qubit partitioning and mapping techniques [16]–[23]. Some work [24]–[26] focuses on tackling the mapping problem in multi-core quantum computers. Additional work incorporates a recent trend and use reinforcement learning for DQC compiling [27], [28]. These approaches are orthogonal to our work. Finally, a hardware-focused body of work recently emerged as well [29]–[32].

III. Proposed Architectural Framework

To motivate our proposed architecture, we first analyze heralded remote entanglement generation for DQC. Then we explain the three principles of our architectural design.

A. Technical considerations of remote entanglement generation

The runtime of DQC in realistic heralded remote entanglement generation is affected by the success probability and the cycle time per attempt. The final fidelity of the DQC circuit output is affected by the quality of the generated entanglement (fidelity of the generated Bell pair). However, entanglement infidelity only injects errors into the circuit locally when implementing remote gates, similar to other noisy local gates.

Success probability per attempt: The success probability of a heralded remote entanglement generation attempt is determined mainly by the following factors:

1) Local photon-qubit entanglement generation probability p_{pq} during one entanglement generation cycle (attempt).
2) Photon transmission loss. For typical low-loss optical fiber the efficiency $\eta_t = \exp(-L/L_{\text{att}})$ is a function of channel length L where the characteristic attenuation length for optical fiber is $L_{\text{att}} \approx 20$ km.
3) In practice photon BSMs are typically implemented using linear optical components whose maximal success probability is 1/2 [33]. In addition, there are other factors such as optical coupling efficiencies and detector efficiencies. We denote the total success probability of the photon BSM as p_s.

Thus, the success probability of one entanglement generation attempt is $p_{\text{succ}} = p_{pq,1}p_{pq,2}\eta_{t,1}\eta_{t,2}p_s \leq 1/2$. Note that both sides must succeed with the transmission.

Cycle time per attempt: The following three processes are the main contributors to the cycle time:

1) For realistic DQC, the allowed waiting time for photon emission should be a fixed value (cutoff time) otherwise the probabilistic nature of emission might lead to an excessively long wait for the emission. On the other hand, the chosen cutoff time will affect the photon emission probability for every attempt of the protocol.
2) If successfully emitted, photons are transmitted via optical channels in which photons travel at the speed of light. The speed of light in optical fiber is 2×10^8m/s. Assuming that the length of the optical fiber from one QPU to the central BSM station is roughly 10m in a DQC center, the on-way transmission time is then $\sim$ 50ns.
3) Latency is introduced when extracting, processing, and transmitting classical measurement results from detectors. Even after obtaining and processing the measurement outcomes, latency

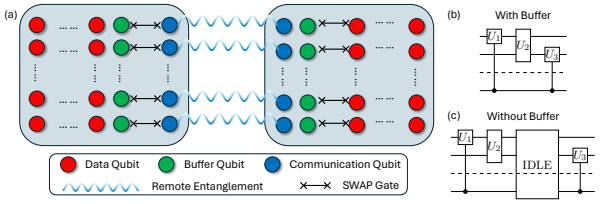


Fig. 2. Schematics of the DQC hardware architecture. (a) Example two-node scenario. (b) Example circuit segment which includes remote gates with buffer qubits. (c) Without a buffer qubit there is additional idling before a remote gate can be implemented due to the wait for entanglement generation.

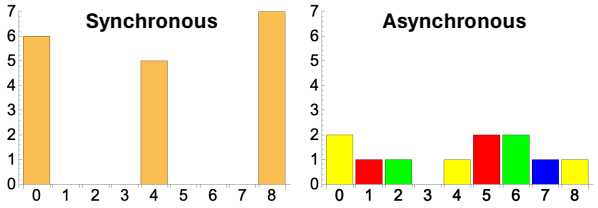


Fig. 3. Example entanglement generation patterns in the time domain. Synchronous and asynchronous attempts between each communication qubit pair are depicted. The vertical axis denotes the number of entangled pairs generated in one time unit, and the horizontal axis denotes time.

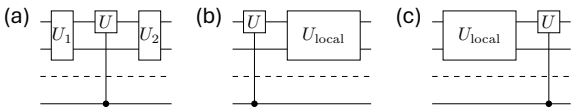


Fig. 4. Example circuit segment variants for different scheduling strategies. (a) Original circuit that does not consider the entanglement generation pattern. (b) ASAP implementation of the remote gate. (c) ALAP implementation of the remote gate.

may be introduced when transmitting the classical feedback using classical communication protocols. The lower bound for this last step is the physical transmission of information at the speed of light.

As a result, the total duration (cycle time) T_{EG} of a single heralded remote entanglement generation attempt is much longer than the cycle times for local operations T_{local}. For the typical case we assume $T_{\mathrm{EG}} \geq 10T_{\mathrm{local}}$ [15].

B. Entanglement buffering

We advocate the use of three types of qubits: buffer qubits, data qubits used to evaluate quantum circuits, and communication qubits which generate entanglement. A schematics of the architecture involving the three different types of qubits is shown in Fig. 2(a). After a pair of communication qubits successfully generates entanglement, each host node applies a local SWAP gate between its communication qubit and a buffer qubit. This results in storing the entangled state in the buffer qubit, whereas the communication qubit is freed up and can continue to be used in entanglement generation attempts. The buffer qubits can store multiple entanglement links to fulfill the demand for remote gates when needed. Decoupling EPR pair generation from storage also enables the pre-initialization of EPR pairs prior to program execution. As demonstrated in Section V, pre-initialized EPR pairs in the buffer (the `init_buf` design) reduce overall circuit latency. We note that QuComm [15] also formalized the concept of buffering in DQC to ensure sufficient communication resources for collective communication. In contrast, our work emphasizes the critical role of buffering in addressing the challenges of probabilistic EPR pair generation and evaluates the effectiveness of buffer qubits in this context. Next we provide additional motivation for introducing buffer qubits with focus on the system design perspective.

Layering: The introduction of buffer qubits in addition to communication qubits and data qubits allows separation of the entanglement generation process in DQC. With buffer qubits, entanglement generation can be implemented as a service that runs continuously [34]–[36] in the background. This hierarchical structure is beneficial for modularization of the DQC hardware architecture, thus simplifying the maintenance of each module. On the other hand, the interaction between communication qubits and buffer qubits to implement the SWAP gate can be realized with quantum interconnect technologies [37], [38].

Multiplexing: By attempting entanglement generation in parallel (multiplexed) using multiple available communication qubits, the latency for receiving an EPR pair is reduced due to the increased effective entanglement generation rate. However, it is unlikely that the entanglement links are generated exactly at the time when the remote gates request them, so they must be stored. If there are no buffer qubits, the communication qubits have to also serve as quantum memories. This means that fewer communication qubits are available and the effective entanglement generation rate decreases. Moreover, an insufficient number of available communication qubits can lead to unnecessary idling due to having no available entanglement links, leading to qubit state degradation and longer runtimes. Such a scenario is illustrated in Fig. 2(b) and (c).

C. Asynchronous remote entanglement generation

The cycle time for remote entanglement generation is generally much longer than the time scale of local quantum operations. For this reason we propose the use of asynchronous entanglement generation among all available communication qubits, a process that is able to "smoothen" the entanglement generation temporal pattern. This yields better resource allocation in the time domain.

In Fig. 3, we visualize the number of successfully generated entanglement links over time, in units of local operation cycle time T_{local}, and without loss of generality we have assumed $T_{\mathrm{EG}} = 4T_{\mathrm{local}}$. In the left panel, when all entanglement generation attempts are synchronous, the entanglement links will arrive in bursts. Remote gates thus need to wait for the burst arrival of entanglement links. After some links in the burst are consumed, the remaining links have to be stored in buffer qubits, which leads to idling decoherence, thus lowering the fidelity of remote gates. In contrast, the right panel demonstrates the pattern of asynchronous attempt when communication qubits are divided into 4 sub-groups, whose starting times of entanglement generation attempts are separated by T_{local}. Different colors denote different sub-groups. In this way, the number of entanglement generated within one time unit is lower, but they distribute more uniformly over time so that some remote gates are able to utilize the entanglement links as soon as they are generated. Additionally, the burst arrival of entangled states can lead to excessive waste when we apply a cut-off policy to buffer qubits, i.e. reset buffer qubits if they store entanglement for too long to avoid too much decoherence in entangled states.

D. Adaptive scheduling of remote gates

Adaptive scheduling of remote gates represents an additional optimization opportunity that can be implemented in software. Note

that for simplicity this work assumes all remote operations are implemented through gate teleportation, and we leave the simultaneous inclusion of both state and gate teleportations as a future work.

Information about existing generated entanglement links can be exploited by the DQC controller. Before a remote gate is executed, we examine whether there are already enough entanglement links stored in the buffer qubits. If yes, we can execute the remote gate earlier so that it can consume existing entanglement as-soon-as-possible (ASAP). This will create a longer time interval between the executed remote gate and subsequent remote gates, and the communication qubits will have a higher chance to create additional entanglement links demanded by the subsequent gates. If there are not enough entanglement links available yet, we can push the remote gate back (as-late-as-possible, ALAP) so that there is more time to generate the required entanglement.

This adaptive circuit modification process must be performed in real time. Because dynamically resynthesizing the quantum circuit based on the number of available EPR pairs e in the buffer is difficult, we can instead statically pre-compile multiple versions of the circuit. To ensure the scalability of the classical compilation, instead of recompiling the whole circuit, we partition the circuit into segments and use a look-up table strategy to manage them. Each segment is compiled with different scheduling policies, and the appropriate circuit version is selected in real time based on the number of available EPR pairs e. To achieve this, we first identify the remote gates and partition the original circuit into segments, where each segment contains m remote gates. The parameter m is tunable and is set to the product of the number of communication qubits and the EPR pair generation rate p_{succ} in our experiments. After partitioning the circuit, we can obtain the variants of the circuit segment corresponding to ASAP and ALAP, respectively, as illustrated by the examples in Fig. 4. During execution, if $e > m$, the controller looks up the next segment with ASAP policy. If $e = 0$, the controller opts for the ALAP policy. Otherwise, the controller uses the original scheduling. We leave more complicated scheduling strategies for future work.

IV. Evaluation Methodology

In this section we describe our performance evaluation benchmarks, system configurations, comparison baselines, and figures of merit. We also describe the entanglement generation dynamics simulation.

A. Benchmarks and Configuration

Benchmarks: We evaluate 6 benchmarks that span different problem sizes and applications. The benchmarks are selected based on the proportion of remote gates in their circuits. The 1D Transverse-Longitudinal Ising Model (TLIM) circuit [39] features linear connectivity and a small number of remote gates. The Quantum Approximate Optimization Algorithm (QAOA) [40] is used to solve the MaxCut problem on regular graphs of degrees 4 and 8, and these circuits therefore have a medium proportion of remote gates. For instance, QAOA-r4-32 represents a 32-qubit QAOA circuit designed to solve the MaxCut problem on a degree-4 regular graph. Finally, the Quantum Fourier Transform (QFT) [41] benchmark requires full connectivity and exhibits a high proportion of remote gates. The benchmark properties are listed in Table I. The code and data that support the findings of this work are available upon request from the authors.

System Configuration: The properties of the different types of quantum operations are listed in Table II. The operation latencies listed in the table agree with those in [15]. In our evaluations, we assume that the success probability of one round of entanglement generation is $p_{\mathrm{succ}} = 0.4$, the backend has a decoherence time $1/\kappa = 150\mu$s where κ is the decoherence rate, and the local CNOT gate time is $300ns$.

TABLE I
Benchmark properties

Name	#qubits	#local 2Q[a]	#remote 2Q[a]	#1Q[a]	depth
TLIM-32	32	300	10	640	40
QAOA-r4-32	32	52	12	64	21
QAOA-r8-32	32	91	34	64	64
QFT-32	32	240	256	32	63
QAOA-r4-64	64	104	28	128	24
QAOA-r8-64	64	174	82	128	84

[a] #local 2Q denotes the number of local two-qubit operations, #remote 2Q denotes the number of remote two-qubit operations, and #1Q denotes the number of single-qubit operations

TABLE II
Quantum Operation Properties

Name	Latency	Fidelity
1Q gates	0.1	99.99%
Local CNOT gates	1	99.9%
Measurement	5	99.8%
EPR pair preparation	10	99%

Baseline: Similar to prior work [42], we utilize the METIS partitioning solver [43] to determine the partitions that are be assigned to different nodes, aiming to minimize the number of remote operations.

B. Figures of merit

We evaluate different designs by comparing the circuit depth and the circuit fidelity. The circuit depth represents the total latency of the circuit. A depth of one corresponds to the latency of a single local CNOT gate. The circuit fidelity represents the quality of the final output. The fidelity is estimated as the product of fidelities of all local single-qubit gates, local two-qubit gates, remote gates implemented through gate teleportation, and an additional idling decoherence factor which accounts for latency. The local gate fidelities we use are listed in Table II. It is important to note that the fidelity of remote gates depends on the fidelity of the consumed entanglement link, which is affected by the buffer qubit decoherence. We use the phenomenological exponential decay factor $\exp(-\kappa t)$, where t is the idling time, to model idling decoherence.

C. Simulation of remote entanglement generation

Inspired by a recent open-source ad hoc quantum network simulation [44], we simulate asynchronous entanglement generation by communication qubits and storage of entanglement by buffer qubits. The system parameters are documented in Sec. IV-A. We assume that the initially generated Bell state is in the Werner form, i.e., a mixture of a pure Bell state and a 2-qubit maximally mixed state. We also assume that all buffer qubits have an identical decoherence rate, and the decoherence channel is the unbiased depolarizing channel, i.e. the three Pauli errors have an identical probability. As a result, the idling dynamics of the Bell state fidelity is given by $F(t) = F_0 \exp(-2\kappa t) + [1 - \exp(-2\kappa t)]/4$, where F_0 is the initial fidelity. The fidelity of a remote gate is obtained through the evaluation of the gate teleportation circuit which includes a noisy Bell state, noisy local 2-qubit gates, and a noisy single-qubit measurement.

V. Results

In this section, we evaluate the DQC circuit depth and fidelity across various architecture designs and system sizes. The designs we consider include `original`: without any buffer qubits, `sync_buf`: with buffer and synchronous EPR pair generation, `async_buf`: with buffer and asynchronous EPR pair generation, `adapt_buf`: with asynchronous EPR pair generation and adaptive remote gate scheduling, `init_buf`: with pre-initiated EPR pairs in buffers, and `ideal`: execution on a monolithic device without remote operations. All reported results represent the average of 50 runs.

A. Comparison of different designs

We evaluated different designs on a simulated 2-node 32-data-qubit DQC architecture with 16 fully-connected data qubits assigned to each node. Each node contains 10 additional communication qubits and 10 buffer qubits. The results are presented in Figure 5 and 6. In Figure 5, we compare the circuit depth across different designs. The y-axis represents the circuit depth relative to the ideal circuit depth. The `original` design, which lacks buffer qubits, results in significant EPR pair waste and a substantial increase of the circuit depth. As shown in the figure, the largest reduction of the depth is achieved by leveraging buffer qubits. The `sync_buf` design reduces the circuit depth by 61.7%. However, the synchronous EPR pair generation pattern in `sync_buf` does not align well with the remote gate request pattern. The asynchronous EPR pair generation design `async_buf` yields an additional average 7% reduction of the circuit depth. Building on this, the adaptive scheduling `adapt_buf` further reduces the depth. By combining the pre-initialized EPR pairs and the adaptive scheduling, `init_buf` achieves an additional 7.5% depth reduction compared to the non-adaptive `async_buf` design.

We also compared the estimated circuit fidelity across different designs. As shown in Figure 6, the `async_buf` and `adapt_buf` designs achieve the same fidelity, yielding an average improvement of 2× and 1.32× compared to the `original` and `sync_buf` designs, respectively. Although the `init_buf` design leverages the pre-initiated EPR pairs to shorten the circuit depth, the extended idle time of these EPR pairs can lead to reduced fidelity.

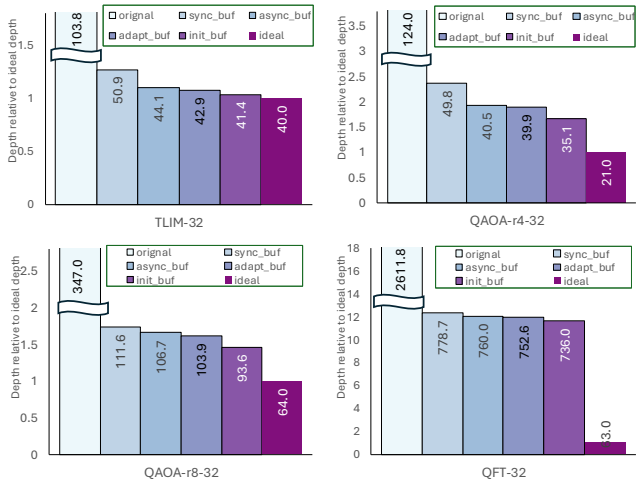


Fig. 5. Comparison of circuit depths across benchmarks and designs.

B. Impact of the number of communication qubits

We evaluate performance with a different number of communication and buffer qubits. The results are shown in Figure 7. We pick the QAOA-r8-32 benchmark since the TLIM benchmark already achieves a near-ideal depth with 10 communication qubits, whereas the QFT-32 benchmark involves an excessive number of remote gates. The `original` design is excluded as it shows minimal change, allowing us to focus on the optimized designs. This also emphasizes that merely increasing the number of communication qubits is ineffective, as EPR pairs are wasted without proper storage in the buffer. As we increase the number of communication and buffer qubits, we notice a significant reduction of the circuit depth. Among the designs, `init_buf` consistently delivers the best performance. When the number of communication qubits reaches 20, it achieves a near-ideal depth, indicating that all remote gates are served immediately. Despite the significant reduction of the circuit depth, the circuit fidelity remains almost unchanged. Hence, we opted to not include the figure. The primary reason is that with asynchronous and adaptive scheduling EPR pairs are consumed immediately after generation, maintaining high fidelity across different cases. The relatively short circuit depth also minimizes the impact of decoherence errors, resulting in negligible differences of the fidelity.

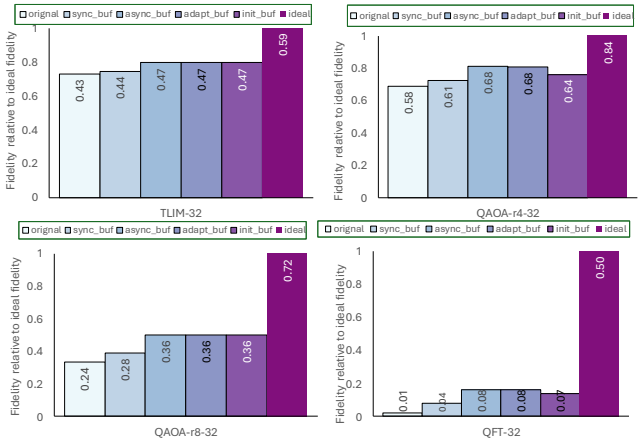


Fig. 6. Comparison of circuit fidelities across benchmarks and designs.

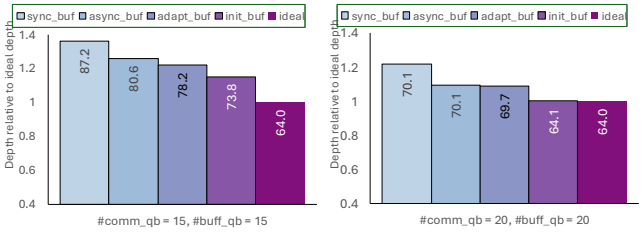


Fig. 7. Circuit depth of QAOA-r8-32 with different numbers of communication and buffer qubits.

C. Evaluations of larger systems

We evaluate two QAOA benchmarks on a 2-node 64-data-qubit system with 32 data qubits allocated to each node. Each node contains 20 additional communication qubits and buffer qubits. The circuit depth comparison is presented in Figure 8. The `init_buf` design achieves a 12% circuit depth reduction compared to the `sync_buf` design. As illustrated, our proposed designs continue to significantly reduce the circuit depth for larger system sizes.

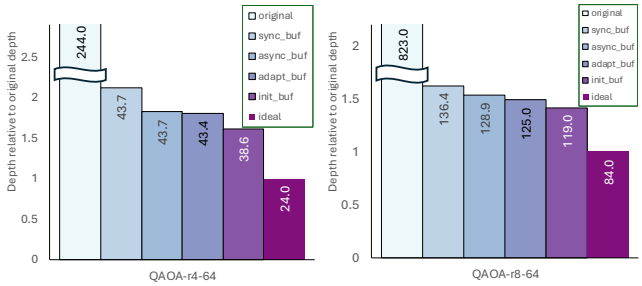


Fig. 8. Comparison of circuit depths across benchmarks and designs on a 64-qubit system.

VI. Conclusion and Discussion

In this paper, we study a new DQC architecture that leverages hardware and software co-design. One of our key observations is that buffering of successfully generated entanglement significantly reduces the circuit latency. Additionally, we introduce an asynchronous entanglement generation strategy that optimizes resource allocation and an adaptive scheduling mechanism that dynamically responds to real-time EPR pair availability. These advancements collectively enhance the performance and scalability of DQC systems, paving the way for more practical and efficient implementations that support the needs of real-world applications.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers. This material is also based upon work supported by the DOE-SC Office of Advanced Scientific Computing Research MACH-Q project under contract number DE-AC02-06CH11357. A.Z. and T.Z. acknowledge support from the NSF Quantum Leap Challenge Institute for Hybrid Quantum Architectures and Networks (NSF Award 2016136).

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, 1999.
[2] D. Cuomo *et al.*, "Towards a distributed quantum computing ecosystem," *IET Quantum Communication*, vol. 1, no. 1, 2020.
[3] M. Caleffi *et al.*, "Distributed quantum computing: a survey," *Computer Networks*, vol. 254, 2024.
[4] D. Barral *et al.*, "Review of distributed quantum computing. From single GPU to high performance quantum computing," *arXiv:2404.01265*, 2024.
[5] A. Carrera Vazquez *et al.*, "Combining quantum processors with real-time classical communication," *Nature*, pp. 1–5, 2024.
[6] F. Afzal *et al.*, "Distributed quantum computing in silicon," *arXiv:2406.01704*, 2024.
[7] D. L. Moehring *et al.*, "Entanglement of single-atom quantum bits at a distance," *Nature*, vol. 449, no. 7158, 2007.
[8] S. Ritter *et al.*, "An elementary quantum network of single atoms in optical cavities," *Nature*, vol. 484, no. 7393, 2012.
[9] J. Hofmann *et al.*, "Heralded entanglement between widely separated atoms," *Science*, vol. 337, no. 6090, 2012.
[10] H. Bernien *et al.*, "Heralded entanglement between solid-state qubits separated by three metres," *Nature*, vol. 497, no. 7447, 2013.
[11] J.-W. Pan *et al.*, "Experimental entanglement swapping: entangling photons that never interacted," *Phys. Rev. Lett.*, vol. 80, no. 18, 1998.
[12] C. H. Bennett *et al.*, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels," *Phys. Rev. Lett.*, vol. 70, no. 13, 1993.
[13] D. Gottesman *et al.*, "Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations," *Nature*, vol. 402, no. 6760, 1999.
[14] A. Wu *et al.*, "AutoComm: A framework for enabling efficient communication in distributed quantum programs," in *IEEE/ACM MICRO*, 2022.
[15] ——, "QuComm: Optimizing collective communication for distributed quantum computing," in *IEEE/ACM MICRO*, 2023.
[16] M. Zomorodi-Moghadam *et al.*, "Optimizing teleportation cost in distributed quantum circuits," *Int. J. Theor. Phys.*, vol. 57, 2018.
[17] P. Andres-Martinez *et al.*, "Automated distribution of quantum circuits via hypergraph partitioning," *Phys. Rev. A*, vol. 100, no. 3, 2019.
[18] Z. Davarzani *et al.*, "A dynamic programming approach for distributing quantum circuits by bipartite graphs," *Quantum Inf. Process.*, vol. 19, 2020.
[19] O. Daei *et al.*, "Optimized quantum circuit partitioning," *Int. J. Theor. Phys.*, vol. 59, no. 12, 2020.
[20] D. Ferrari *et al.*, "Compiler design for distributed quantum computing," *IEEE TQE*, vol. 2, 2021.
[21] S. DiAdamo *et al.*, "Distributed quantum computing and network control for accelerated VQE," *IEEE TQE*, vol. 2, 2021.
[22] D. Dadkhah *et al.*, "Reordering and partitioning of distributed quantum circuits," *IEEE Access*, vol. 10, 2022.
[23] D. Ferrari *et al.*, "A modular quantum compilation framework for distributed quantum computing," *IEEE TQE*, 2023.
[24] J. M. Baker *et al.*, "Time-sliced quantum circuit partitioning for modular architectures," in *ACM CF*, 2020.
[25] M. Bandic *et al.*, "Mapping quantum circuits to modular architectures with QUBO," in *IEEE QCE*, 2023.
[26] P. Escofet *et al.*, "Revisiting the mapping of quantum circuits: Entering the multi-core era," *ACM TQC*, 2024.
[27] P. Promponas *et al.*, "Compiler for distributed quantum computing: a reinforcement learning approach," *arXiv:2404.17077*, 2024.
[28] E. Russo *et al.*, "Attention-based deep reinforcement learning for qubit allocation in modular quantum architectures," *arXiv:2406.11452*, 2024.
[29] J. Ang *et al.*, "ARQUIN: Architectures for multinode superconducting quantum computers," *ACM TQC*, vol. 5, no. 3, 2024.
[30] J. Kim *et al.*, "A fault-tolerant million qubit-scale distributed quantum computer," in *ACM ASPLOS*, 2024.
[31] C. Chu *et al.*, "TITAN: A fast and distributed large-scale trapped-ion NISQ computer," in *ACM/IEEE DAC*, 2024.
[32] S. Bahrani *et al.*, "Resource management and circuit scheduling for distributed quantum computing interconnect networks," *arXiv:2409.12675*, 2024.
[33] N. Lütkenhaus *et al.*, "Bell measurements for teleportation," *Phys. Rev. A*, vol. 59, no. 5, 1999.
[34] A. Kolar *et al.*, "Adaptive, continuous entanglement generation for quantum networks," in *IEEE INFOCOM Workshops*, 2022.
[35] A. Zang *et al.*, "Analytical performance estimations for quantum repeater network scenarios," in *IEEE QCE*, 2024.
[36] C. Zhan *et al.*, "Design and simulation of the adaptive continuous entanglement generation protocol," *arXiv:2502.01964*, 2025.
[37] D. Awschalom *et al.*, "Development of quantum interconnects (QuICs) for next-generation information technologies," *PRX Quantum*, vol. 2, no. 1, 2021.
[38] ——, "A roadmap for quantum interconnects," Argonne National Laboratory, Tech. Rep., 2022.
[39] A. Sopena *et al.*, "Simulating quench dynamics on a digital quantum computer with data-driven error mitigation," *Quantum Science and Technology*, vol. 6, no. 4, 2021.
[40] E. Farhi *et al.*, "A quantum approximate optimization algorithm," *arXiv:1411.4028*, 2014.
[41] D. Coppersmith, "An approximate Fourier transform useful in quantum factoring," *quant-ph/0201067*, 2002.
[42] M. G. Davis *et al.*, "Towards distributed quantum computing by qubit and gate graph partitioning techniques," in *IEEE QCE*, 2023.
[43] G. Karypis *et al.*, "METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.
[44] A. Zang *et al.*, "Entanglement distribution in quantum repeater with purification and optimized buffer time," in *IEEE INFOCOM Workshops*, 2023.

PipeLink: A Pipelined Resource Sharing System for Dataflow High-Level Synthesis

Rui Li
Intel
rui.lee92@gmail.com

Lincoln Berkley
Yale University
linc.berkley@yale.edu

Rajit Manohar
Yale University
rajit.manohar@yale.edu

***Abstract*—Dynamically scheduled high-level synthesis (HLS) is an approach to HLS that maps programs into dataflow circuits. These circuits use distributed control for communication and therefore can be automatically pipelined. However, pipelined function call is challenging due to the absence of centralized control, thus general software programs cannot be well-supported by the HLS tools. Traditional solutions to this problem impose restrictions on pipelining when accessing the shared functions. We present PipeLink, a modular synthesis method that decomposes the synthesis into compilation stage and linking stage, which enables the fully-pipelined access to shared functions in dataflow circuits. We develop a complete HLS engine using this approach and use asynchronous circuits as the target implementation. Our engine supports pipelined access to shared function units, as well as pipelined memory access in a unified fashion. Compared to existing HLS tools, PipeLink results in 20X reduction in energy and 1.56X improvement in throughput.**

I. Introduction

High-level synthesis (HLS) is a design methodology that automatically maps software programs (e.g., C/C++) into a hardware description language that implements the same functionality. It has received a lot of interests in hardware accelerator design [26], [34] since it can reduce design time and is more accessible to software developers. There are two major categories of HLS: statically scheduled HLS (static HLS) and dynamically scheduled HLS. Statically scheduled HLS [17], [27] converts a program into a datapath that contains all the operators from the program and a global state machine that enforces centralized control for these operators. To synthesize the state machine, the static HLS tool needs accurate delay information (in terms of clock cycles) for all operators at synthesis time and uses this information to compute an optimized schedule. This early scheduling may lead to sub-optimal performance, but the central scheduler design makes it good at coordinating different circuit components and managing resource access conflicts.

Dynamically scheduled HLS [14], [23], [24], [28], [33], [38], on the other hand, maps programs into dataflow circuits consisting of concurrent, independent dataflow elements. The dataflow elements communicate with each other through local control (i.e., handshake or ready/valid protocols), so they can be automatically pipelined without requiring global control. However, the lack of global control makes it hard to coordinate the actions of different dataflow elements.

The research in this paper was supported in part by DARPA IDEA grant FA8650-18-2-7850, and in part by DARPA POSH grant HR001117S0054-FP-042.

In this work, we argue that currently both static HLS and dynamically scheduled HLS have limitations when faced with the combination of pipelined design and general hardware resource sharing. To address this issue, we propose the modular synthesis methodology, which decomposes the synthesis into compilation and linking stages, each of which captures different aspects of program structures. We show how this approach enables the pipelined resource sharing in dataflow circuits, and permits additional decoupling between control flow and data flow in the hardware, overcoming the limitations of previous approaches.

We implement our proposed techniques in PipeLink, an HLS engine that enables pipelined resource sharing in dataflow circuits. PipeLink makes a number of contributions to the dynamically scheduled HLS: (i) it proposes a modular method for synthesizing dataflow circuits, which enables fully-pipelined resource sharing in dataflow circuits; (ii) it supports both pipelined function calls and pipelined memory accesses in a unified design; (iii) it directly synthesizes efficient circuits on unmodified programs (i.e., without requiring pragmas being inserted by users). We evaluate our HLS approach against both academic and commercial HLS tools. Overall, PipeLink provides significant improvement in energy, some improvement in latency and throughput while paying some area penalty.

The remaining paper is organized as follows: Section II introduces the dataflow circuits as well as the requirements and challenges for sharing the resources at circuit level. It also discusses various options of sharing the resources and the pros and cons of existing techniques. Section III introduces PipeLink and how it leverages the modular synthesis method to enable pipelined resource sharing that is not available previously. Section IV shows the evaluation results.

II. Background

A. Dataflow circuits

Dataflow circuits consist of concurrent, independent dataflow elements. Each dataflow element communicates with the predecessors and successors through local handshake protocols, thus it does not require global control. A dataflow element remains idle until it receives input tokens from its predecessors [18]. This is different from static HLS tools, where all the building blocks require a global state machine to coordinate.

Consider the *if* example in Fig. 1 and the synthesized dataflow circuit. In the true branch, x is conditionally used

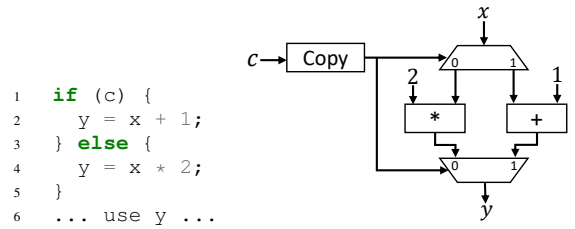


Fig. 1: dataflow circuit example

```
y0 = f(x0);
...
y1 = f(x1);
```
a) two invocations to f

```
y0 = f(x0);
y1 = f(x1);
```
b) sequential code

```
y0 = f(x0);
if (c) {
  y1 = f(x1);
}
```
c) conditional code

```
y0 = f(x0);
do {
  y1 = f(x1);
} while (c);
```
d) loop code

Fig. 2: Example code for function call

and y is redefined. In the false branch, nothing happens. After the if statement, y is used. The synthesized circuit has two types of operators: $MERGE$ and $SPLIT$ for circuit control. $MERGE$ receives multiple inputs and sends one of them to the output port. $SPLIT$ receives one input and sends it to one of the multiple output ports. The selection is controlled by the token received from the control port. For both operators, the left (right) port is selected under the false (true) condition. A complete set of dataflow components also includes token source, token sink, copy, function computation component, and an initial token buffer [36].

As pointed out by existing works [14], [16], [23], [28], [36], [38], dataflow circuits can be automatically pipelined. In this example, x is sent to the correct $FUNC$ based on c, and the $FUNC$ will send its result to the $MERGE$ immediately when the computation is done; the whole process is data-driven, without requiring centralized (global) control.

B. Resource sharing problem for HLS

In software programs, function calls and memory accesses (from explicitly declared arrays and dynamically allocated memories) are pervasive, and the corresponding functions and memories are shared among different callers. An HLS tool has the choice of inlining functions at each call site, or having a a single hardware block for the function that is shared by the call sites, or some combination. However, memory banks/global memory has to be viewed as a shared resource.

For the purposes of studying the resource sharing problem, a memory can be viewed in the same way as a function. A memory has input ports for address, read/write command, and data (used for write operations), and an output port that produces data (used for read operations), so we can treat memory access as a special type of function call where the function cannot be inlined/replicated. In this paper, we use "function" to denote both regular functions and memory banks, and our system supports both in a unified way. Note that the "function" refers to the regular software function, not necessarily the combinational function unit.

We are not arguing that resource sharing is the optimal solution to handle function calls and memory accesses; instead, other options such as function inlining can be used as well. However, this work focuses on pipelined resource sharing problem, so we illustrate the resource sharing design for function calls and memory accesses. It is an open question about the right balance between resource sharing and function inlining, which is out of the scope of this paper.

To start with, Fig. 2(a) shows a program segment where `f` is invoked twice. Suppose `f` is a single input, single output function, and it has two callers `y0` and `y1`. Let's annotate the two invocations to `f` as f^0 and f^1 respectively. Furthermore, let's assume `f` is not inlined. `f` is synthesized as an independent hardware block, and its in/output ports as well as `x0`, `x1`, `y0` and `y1` are synthesized as dangling ports, as illustrated in Fig. 3(a). The HLS tool needs to synthesize a resource sharing system to send `x0` and `x1` to `f` and propagates the result from `f` to `y0` and `y1` respectively.

The access order to `f` is important. Each invocation could have side-effects: `f` could modify memory (Fig. 4(a)), or update static variables (Fig. 4(b)). To guarantee correctness, it is important to make sure `f` is accessed in the same order as the original software program.

Furthermore, the number of accesses to `f` for each invocation is unknown during static synthesis stage. Instead, it needs to be dynamically computed based on runtime conditions. For example, Fig. 2(a) shows a code segment of two invocations to `f`. For sequential invocation (Fig. 2(b)), f^0 and f^1 each access `f` once. For conditional invocation (Fig. 2(c)) where c is false during a specific run, f^1 should not access `f`. For iterative invocation (Fig. 2(d)) where the loop has N iterations during a specific run, f^1 should access `f` N times before the next invocation of `f`.

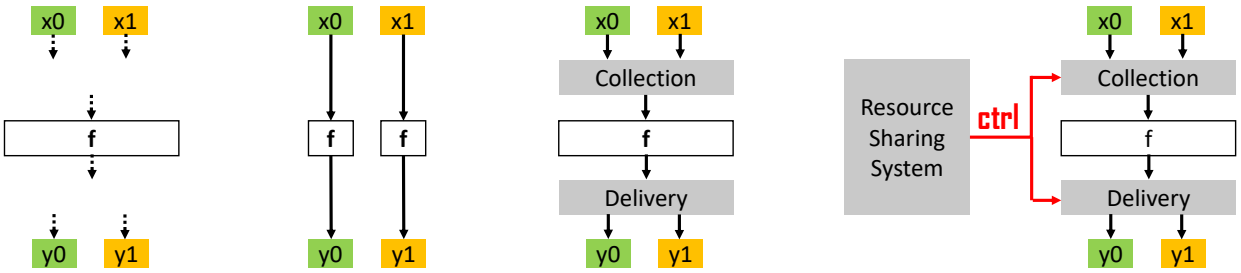
To summarize, the resource sharing system needs to:

1) enforce the **access order** to `f` to be the same as specified by the programs; for example, `x0` should be sent to `f` before `x1` (Fig. 4(a));
2) count the **number of accesses** to `f` for each invocation, and starts the next invocation only when the current one has finished using `f`;

Note that because the original software program is sequential, all of the information necessary to determine the ordering and access count is available at run-time; the challenge is to compute it using dataflow operators.

C. Resource sharing options

The general resource sharing problem is illustrated in Fig. 3(a), where `f` is shared at two independent call sites. We assume `f` is a single-in-single-out resource block, x_0 and x_1 are the input tokens at two call sites, and y_0 and y_1 are the receivers to collect the results at each call site. Current HLS tools are mainly optimized for efficient synthesis of a single function, but they fall short in handling generic function calls in software programs.



a) resource sharing problem b) replication c) blocking access d) pipelined access

Fig. 3: *Different resource sharing options*

```
1  int A[M];
2  int f(int i) {
3    A[i]++;
4    ...
5  }
```

a) modify memory

```
1  int f(int i) {
2    static int count = 0;
3    count++;
4    ...
5  }
```

b) update static variable

Fig. 4: Side effects of the function invocation

1) Replication

If `f` is inlined, then the resource sharing problem is automatically solved by replicating the resources of `f` at each call site, as shown in Fig. 3(b). This is the default options for the commercial HLS tools [5], [9], [10] as well as academic HLS tools [15], [30], [32]. This solution can dramatically increase the circuit cost if the shared functions are expensive. Furthermore, this approach cannot be used for memories that are accessed from more than one program location. Some commercial tools allow users to encapsulate functions as operators and then decide how many of these to instantiate, which controls the amount of parallelism to be extracted. However, this option is not feasible when the functions have side-effects as shown in Fig. 4.

2) Blocking sharing

As shown in Fig. 3(c), the HLS engine should synthesize a *collection* circuit to collect and select the arguments to `f` from all callers, as well as a *delivery* circuit to deliver the results back to the appropriate caller. In the blocking approach, there is a lack of explicit *control* to coordinate the access order to `f`; hence, access to `f` is granted on a first-come first-served basis. However, the order in which access to `f` is requested *must match the software program order* whenever `f` has side-effects (e.g., Fig. 4). This means that if the underlying HLS implementation has any pipelining, extra logic is needed to guarantee that this access ordering is preserved. This applies to both accesses from different call sites, as well as repeated accesses from a fixed call site. Without extra logic, this would violate the requirements for accessing the shared resource.

To handle both these issues, existing solutions require that `f` is shared in a blocking fashion, i.e., `x1` cannot be generated and sent to `f` until `f` receives `x0` and produces `y0`. An alternate approach is to only have blocking on the collection logic end, and generate a control token in the collection logic that is used to route the results to the appropriate invocation [23]. For example, if `x0` is selected, then a control token `0` can be generated by the collection circuit and routed to the delivery logic, which then can route the result to `y0`; the *ARBITER* dataflow element can be used for this purpose. However, access to the collection circuit is still blocking. In particular, in the pipelined context, the next `x0` invocation must block until all possible future invocations from the current execution have been considered. Furthermore, using an *ARBITER* means that slack elasticity is no longer guaranteed, and hence the slack elascity feature of the circuits cannot be guaranteed.

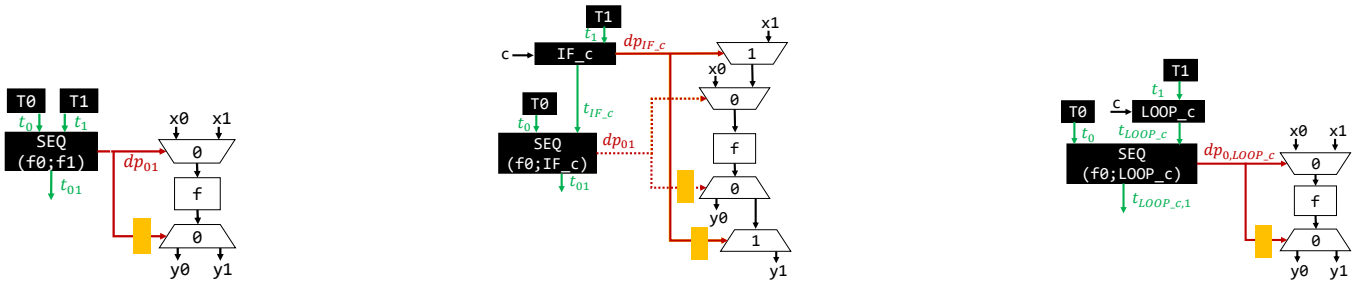
3) Pipelined sharing

Fig. 3(d) shows the pipelined resource sharing system for `f`, which explicitly calculates the *ctrl* tokens required. The benefit of this approach is that the control generation is *decoupled* from the data tokens, and often can be computed earlier since it only depends on the control flow of the program.

Static HLS solution. Static HLS generates a global state machine to provide central control to all operations in the circuits, so it can easily generate the desired *ctrl* tokens for the resource sharing system. However, in order to generate a pipelined hardware block for `f`, it is essential to decide the *initial interval (II)* for `f` during static synthesis stage. To do that, it is often required to inline the sub-functions and unroll all the loops inside `f` [1], [6], [10], which can result in high circuit overhead. In addition, static HLS approaches do not optimize for two common cases: (1) Conditional executions in `f`. Static HLS has to consider the worst-case scenario for synthesizing `f`. If `f` has nested, unbalanced *if/switch* statements, this problem gets worse. (2) Early-return in `f`: Sometimes a function will return in the middle of the function body for fast computation. However, the static HLS has to consider the whole function body when calculating the II, completely ignoring this shortcut.

In practice, pipelined resource sharing in static HLS is only enabled under user's discretion (typically via `#pragma` directives in C/C++) due to its potential high overhead.

Dynamically scheduled HLS solution. The dynamically scheduled HLS does not synthesize the global state machine to schedule each operator in cycles, which makes it very hard to generate the desired *ctrl* tokens to coordinate the access to the shared resource blocks. To handle this problem, we argue that the circuit synthesis should be decoupled into two stages. In the first compilation stage, each software function should be



a) *SEQ* b) IF_c c) $LOOP_c$

Fig. 5: Token network template

mapped into circuits independently, leaving the function calls and memory accesses unprocessed; in the second linking stage, the complete function call graph structure is analyzed, and the corresponding linking circuit is synthesized to connect the shared resources to the call sites. To our knowledge, we present the first solution to the problem of fully pipelined resource sharing for dataflow circuits that are generated from programs. Our solution results in a purely deterministic dataflow circuit implementation, ensuring slack elasticity. As a result, our approach addresses the control token generation problem in the general case [12] and supports pipelined resource sharing for dynamically scheduled HLS.

III. Moduler Synthesis Methodology

PipeLink decomposes the synthesis into two steps. During compilation step, each function is synthesized into a distinct resource block. If there are accesses to the shared resources, then the input tokens to the shared resources, together with the receivers to collect results from the shared resources, are synthesized as dangling ports, as shown in Fig. 3(a). During the linking step, PipeLink would analyze the program structure where the function call happens, and generate the control circuit to connect the dangling ports (synthesized in the compilation stage) following the program structure. The control circuit, called *Adaptive Control Token Network (ACTN)*, is itself a pipelined dataflow circuit that correctly computes the control tokens needed for pipelined resource sharing (as shown in Fig. 3(d)).

As mentioned in Section II-B, ACTN needs to count the **number of accesses** to `f` for each call site, and enforce the **access order** to the shared resource `f` as specified in programs. To achieve these, it captures the control structure of the program as well as runtime conditions to compute the control tokens. First, it generates a *use-resource* sequence for each invocation to serve as its private control token stream and resource access counter to count the number of accesses, so different invocations are decoupled from each other and do not need to compete for the single shared control token. Second, it composes different use-resource sequences to control the access order to the shared function `f`. It provides additional decoupling between control flow and data flow, reducing the chance of them blocking each other.

A. The "use-resource" sequence

A use-resource sequence is generated for each invocation to `f`. It encodes the number of accesses to `f` for the invocation during current run. It is a one-bit token stream whose value is run-length encoded. For example, if the invocation f^0 accesses `f` three times, the use-resource sequence would be `1,1,1,0`. Each "`1`" enables f^0 to access `f` once, and `0` represents the end of accessing `f` from the specific call site corresponding to the use-resource sequence. We now describe how we construct this sequence from the structure of the program.

B. The base case: a single function call

The base case corresponds to a single function call invocation. The use-resource sequence for this case is simply `1,0`, meaning this invocation would access `f` exactly once when the program reaches this call site.

C. Composing use-resource sequences

ACTN composes use-resource sequences (corresponding to all call sites of `f`) together by structural induction on the program to generate the final use-resource sequence and control tokens needed for resource sharing. There are three types: sequential, conditional and iterative.

Sequential composition: *SEQ*. Consider the example in Fig. 2(b) where two programs are composed in a sequential fashion and both include one invocation to `f`. As shown in Fig. 5(a), there have two use-resources sequences, t_0 and t_1, generated for the two invocations. Since we are constructing this by structural induction, the sequences t_0 and t_1 have variable length that in general is only known at run-time.

To compose the two programs: (a) we use *MERGE* / *SPLIT* to route the arguments/results of `f` from/to the dataflow circuits. (b) we compute the final use-resource sequence t_{01} from t_0 and t_1. (c) we generate a control token sequence dp_{01} for the *MERGE* and *SPLIT* to route the datapath tokens.

SEQ network generates t_{01} as follows: `1`s from t_0 are read and propagated to the output; when a `0` is encountered, *SEQ* switches to t_1 and continues forwarding `1`s from t_1 to the output until a `0` is encountered. Then, *SEQ* forwards the `0` to the output, and switches back to t_0. If we run the code segment in Fig. 2(b) once, t_0 is `1,0`, t_1 is `1,0` and t_{01} is `1,1,0`. dp_{01} sequence is generated as follows: for each `1` from t_0, a `0` is generated; for each `1` from t_1, a `1` is generated. The final dp_{01} is `0,1` for our example.

Conditional composition: *IF_c*. If `f` is called in a conditional statement where c is the condition, then the use-resource sequence can be computed using c. Suppose we have the following program `if(c) {S} else {T}`, and t_S and t_T

TABLE I: Normalized Performance (compared to Legup)

	PipeLink				*Commercial 1*				*Commercial 2*			
Benchmark	**Delay**	**Area**	**Energy**	**Throu.**	**Delay**	**Area**	**Energy**	**Throu.**	**Delay**	**Area**	**Energy**	**Throu.**
dfmul	0.51	1.50	0.04	1.99	1.03	0.69	0.70	0.97	1.00	1.23	0.66	1.00
dfdiv	0.53	1.40	0.01	1.89	0.47	0.88	0.39	2.11	0.72	1.93	1.67	1.39
sha	0.61	1.27	0.10	1.63	0.94	0.99	1.08	1.06	1.02	0.98	1.18	0.98
adpcm	0.49	1.29	0.02	2.08	0.86	0.55	0.82	1.17	0.75	1.04	1.16	1.33
diff	0.91	0.58	0.08	1.10	0.61	0.49	0.41	1.64	1.02	1.42	1.11	0.98
fft	0.74	1.16	0.11	1.39	0.86	0.78	0.63	1.16	0.98	0.93	0.79	1.02
fir	0.88	0.72	0.15	1.13	0.94	0.78	0.90	1.06	1.00	0.93	1.01	1.00
geomean	0.65	1.08	0.05	1.56	0.79	0.72	0.56	1.26	0.92	1.17	1.04	1.09

are the use-resource sequences for the corresponding branches. The final use-resource sequence is t_{if}, and the datapath control token sequence is dp_{if}.

If c is false we propagate `1`s from t_T to t_{if} and produces `0`s for dp_{if} until `0` is encountered. If c is true, we propagate `1`s from t_S to t_{if} and produces `1`s for dp_{if} until `0` is encountered. In the end, we appends a `0` to the end of t_{if}.

In Fig. 2(c), `f` is invoked twice: f^1 (Line 3) executed in a if block IF_C, and f^0 (Line 1) and IF_C are executed sequentially. Fig. 5(b) shows the synthesized circuits. Since f^1 is conditionally invoked, $MERGE_1$ and $SPLIT_1$ are synthesized to conditionally send argument x_1 to `f` and collect result for y_1, which are controlled by the IF_c network. In general IF_c network takes in three inputs: if condition c, and two `use-resource` tokens for both branches. (Note that we have omitted the constant "0" use-resource stream as an input to IF_c, since `f` is only used in the true branch.)

Iterative composition: *LOOP_c*. Suppose the use-resource sequence for the loop body is t_{body}, and the final use-resource sequence is t_{loop}. While loop condition is true, the `1`s from t_{body} are propagated to t_{loop}. When the loop terminates, we append a `0` to t_{loop}. The net effect is: t_{loop} accumulates the `1`s from t_{body} in all iterations. Since the loop has a single body, no $MERGE$/$SPLIT$ or control token generation is needed. In Fig. 2(d), `f` is invoked twice: f^0 (Line 1) executed in a sequential block, and f^1 (Line 3) executed in a loop block $LOOP_C$. Fig. 5(c) shows the synthesized circuits.

D. Summary

ACTN is dynamically constructed based on the induction of program structures using pre-designed template circuits. It operates in parallel with the datapath to decouple the control path and the data path. To reduce stalls in the use-resource network, FIFO buffers are inserted at the delivery circuit end (shown as yellow blocks in Fig. 5). As a result, new datapath control tokens can be generated (thus enabling new invocations to start) without waiting for the previous invocation to finish.

IV. Evaluation

We implement PipeLink in LLVM [7]. First, optimized LLVM IR is generated with *clang* and existing optimization passes. Next, we implement the ACTN (Section III) as LLVM passes [8]. These passes accept optimized LLVM IR as input and generate optimized dataflow circuit graphs as their output. They also perform operator clustering optimizations to group the combinational operators together for logic optimization. Finally, we build on the asynchronous circuit back-end developed by [28] to map the optimized dataflow graphs into asynchronous circuits in a 28nm technology.

A. Setup

Circuit synthesis. Each dataflow graph component is translated into a unique pipeline stage, and the data transfers between pipelined stages use the bundled data protocol [29]. The control for each pipelined stage uses micro-pipelines [35]. The control circuits are custom designed asynchronous logic, and the bundled-data datapath uses combinational logic for computation that is mapped to a standard cell library.

Simulation methodology. We built a discrete-event simulator to simulate the synthesized bundled data circuits with 4-phase handshake for process communication. We use HSPICE to extract the control circuit performance metrics, and commercial logic synthesis tool to determine delay/power/area of combinational logic, both under a 28nm process technology. Synchronous results were obtained using the same cell library and logic synthesis tool.

Comparison. We use four applications (*dfmul, dfdiv, sha, adpcm*) from an HLS benchmark suite [3], [21], and three widely used benchmarks: a differential equation solver (*differential*) from [31], and *fft* and *fir* from [15]. We compare against an academic HLS tool *LegUp* v4.0 [15] as well as two commercial HLS tools *Commercial 1* and *Commercial 2*. We use default options for all of the HLS tools and do not insert any pragmas in the benchmarks.

Metrics. We measure Delay (ps), Area (μm^2), Energy (pJ) and Throughput (MHz). We do not show leakage power results, as they are always proportional to the area. We run each benchmark twenty times (the input data are provided by each benchmark, and there are dozens of them to cover different cases) and use averages across the runs to report benchmark statistics. We use the same methodology to collect results for both PipeLink and other HLS tools.

B. Experiment results

Comparison. Table I shows the normalized performance numbers of PipeLink, *Commercial 1* and *Commercial 2* compared to *Legup*. Fig. 6 shows per-benchmark spider plots of normalized performance (relative to Legup) as well as the geometric mean of the normalized performance across all HLS benchmarks. Note that we plot the inverse of the normalized throughput; hence, for all metrics, lower is better. These spider plots can better illustrate the trade-off among Delay, Area, Energy and Throughput.

PipeLink consistently has the best energy per benchmark. Compared with Legup, the minimum energy saving is $6.67X$ for *fir*, the maximum saving is $100X$ for *dfdiv* and the average

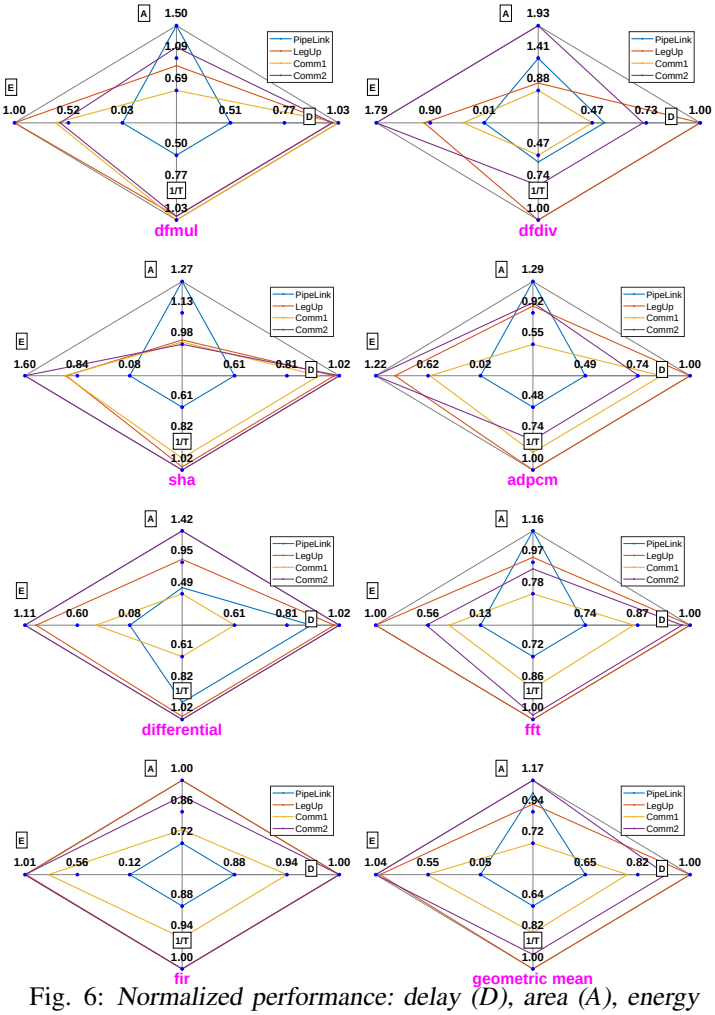


Fig. 6: *Normalized performance: delay (D), area (A), energy (E), leakage power (L), and throughput-inv (1/T)*

saving is $20X$. Even compare with *Commercial 1*, PipeLink still saves energy by $12X$ on average. PipeLink synthesizes asynchronous circuits and only triggers the processes with input data tokens, which results in significant energy savings.

In most cases, PipeLink has the smallest delay ($0.62X$ on average) and the highest throughput ($1.64X$ on average). The improvement comes from two aspects: (i) PipeLink does extensive operator clustering, which results in lower latency circuits through logic optimization and specialization; (ii) when there are (nested-)*if* statements, the delay/throughput of asynchronous circuits depends on the activated processes at run-time while synchronous circuits are limited by the worse-case scenario. For *differential* benchmark, *Commercial 1* ($0.61X$) is much faster than PipeLink ($0.91X$) because *differential* primarily consists of a loop. PipeLink has to synthesize multiple $MERGE$s and $SPLIT$s for loop control [28], [36], and they increase the total delay and lower the throughput.

PipeLink increases the area by $1.61X$ on average. The penalty can be attributed to the synthesis of $MERGE$s and $SPLIT$s for *if* and *loop* statements that remain unfused, and the collection and delivery circuits for accessing shared resources. In addition, PipeLink does not currently include an operator scheduling phase, and hence datapath operators that might be shared by commercial HLS tools would be replicated in PipeLink.

V. Related work

Dynamically scheduled HLS. Some synchronous HLS tools generate dataflow circuits [23], [24], [37]. Lana et al. have done a series of works on mapping C/C++ programs into synchronous dataflow circuits, including [22]–[25]. DynSchedule [23], GeneralCodeSynthesis [24] and CToDataflow [25] use the distributed token network design for pipelined memory access, but the circuit pipelining could be harmed as described in Section II-C2. Furthermore, they rely on customized out-of-order LSQ [22] for pipelined memory accesses within the activated basic block in program's control flow graph, which could potentially incur high overhead. Lastly, there is not much descriptions about how they support general-purpose function calls in a pipelined manner. PipeLink however, unifies the pipelined function calls and pipelined memory accesses, and proposes an adaptive control token network design to improve circuit pipelining over distributed token network design. FunctionSynthesis [37] synthesizes functional programs into pipelined dataflow circuits. In functional programming, each function is "stateless", so their stateless functions can be shared in a pipelined manner without requiring to enforce the access order among invocations. Some tools directly map software programs into asynchronous dataflow circuits. Spatial computing [14], CASH [38] and Pegasus [13] generates asynchronous circuits which are based on the MicroPipelines [35] design, and they only support blocking function calls. AutoCtrl [20] generates distributed asynchronous control circuits automatically, but it only supports blocking function calls.

Asynchronous synthesis. Balsa [19] is based on syntax-directed translation to map message-passing hardware description language (HDL) into an asynchronous circuit, and it only allows sequential function calls. STF [36] and DataDecomposition [39] maps high-level circuit description programs into concurrent hardware modules with restricted support of pipelined function calls.

Static HLS. Most commercial tools are static HLS tools [1], [2], [4], [5], [10], and they will do function inlining by default when doing the synthesis. They also provide directives and pragmas [1], [6], [11] for users to explicitly specify function pipelining. However, pipelined resource sharing in statically-scheduled circuits could be costly for unrolling the loops and inlining functions (Section II-C3), so users need to be very careful before deciding to pipeline a function. The academia HLS tools [15], [30], [32] also share similar problems.

VI. Conclusion

We propose PipeLink, a dynamically scheduled HLS engine to map programs into asynchronous dataflow circuits. It proposes a modular synthesis methodology to support pipelined function calls and pipelined memory access uniformly. Results show that PipeLink can dramatically save energy, and improve delay and throughput performance.

REFERENCES

[1] Cadence hls. https://sen.enst.fr/system/files/c8927325ebae2ddae9620055f8a2396c/cadence_ctos_user_guide.pdf.

[2] Catapult C. https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/hls/c-cplus.

[3] Chstone. https://github.com/A-T-Kristensen/patmos_HLS/tree/master/benchmarks/CHStone.

[4] Intel HLS. https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html.

[5] Legup. https://www.legupcomputing.com/.

[6] Legup directive. https://download-soc.microsemi.com/FPGA/HLS-EAP/docs/legup-9.1-docs/index.html/.

[7] LLVM. https://llvm.org/.

[8] LLVM optimization pass. https://llvm.org/docs/Passes.html.

[9] Stratus. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/stratus-high-level-synthesis.html.

[10] Vivado hls. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug902-vivado-high-level-synthesis.pdf.

[11] Vivado hls directive. https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/oxu1504034430931.html.

[12] Arvind. Dataflow: passing the token. Keynote address, International Symposium on Computer Architecture (ISCA) (http://csg.csail.mit.edu/Users/arvind/ISCAfinal.pdf).

[13] Mihai Budiu and Seth Copen Goldstein. Pegasus: An efficient intermediate representation. Technical report, Carnegie Mellon University, 2002.

[14] Mihai Budiu, Girish Venkataramani, Tiberiu Chelcea, and Seth Copen Goldstein. Spatial computation. In Shubu Mukherjee and Kathryn S. McKinley, editors, *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2004, Boston, MA, USA, October 7-13, 2004*, pages 14–26. ACM, 2004.

[15] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason Helge Anderson, Stephen Dean Brown, and Tomasz S. Czajkowski. Legup: high-level synthesis for fpga-based processor/accelerator systems. In John Wawrzynek and Katherine Compton, editors, *Proceedings of the ACM/SIGDA 19th International Symposium on Field Programmable Gate Arrays, FPGA 2011, Monterey, California, USA, February 27, March 1, 2011*, pages 33–36. ACM, 2011.

[16] Jianyi Cheng, Lana Josipovic, George A. Constantinides, Paolo Ienne, and John Wickerson. Combining dynamic & static scheduling in high-level synthesis. In Stephen Neuendorffer and Lesley Shannon, editors, *FPGA '20: The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23-25, 2020*, pages 288–298. ACM, 2020.

[17] Philippe Coussy and Adam Morawiec. *High-level synthesis*, volume 1. Springer, 2010.

[18] Jack B. Dennis. The evolution of 'static' data-flow architecture. In J.-L. Gaudiot and L. Bic, editors, *Advanced Topics in Data-Flow Computing*. Prentice-Hall, 1991.

[19] Doug A. Edwards and Andrew Bardsley. Balsa: An asynchronous hardware synthesis language. *Comput. J.*, 45(1):12–18, 2002.

[20] Euiseok Eim, Jeong-Gun Lee, and Dong-Ik Lee. Automatic process-oriented control circuit generation for asynchronous high-level synthesis. In *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000) (Cat. No. PR00586)*, pages 104–113, 2000.

[21] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, Hiroaki Takada, and Katsuya Ishii. Chstone: A benchmark program suite for practical c-based high-level synthesis. In *International Symposium on Circuits and Systems (ISCAS 2008), 18-21 May 2008, Sheraton Seattle Hotel, Seattle, Washington, USA*, pages 1192–1195. IEEE, 2008.

[22] Lana Josipovic, Philip Brisk, and Paolo Ienne. An out-of-order load-store queue for spatial computing. *ACM Trans. Embed. Comput. Syst.*, 16(5s):125:1–125:19, 2017.

[23] Lana Josipovic, Andrea Guerrieri, and Paolo Ienne. Invited tutorial: Dynamatic: From C/C++ to dynamically scheduled circuits. In Stephen Neuendorffer and Lesley Shannon, editors, *FPGA '20: The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23-25, 2020*, pages 1–10. ACM, 2020.

[24] Lana Josipovic, Andrea Guerrieri, and Paolo Ienne. Synthesizing general-purpose code into dynamically scheduled circuits. *IEEE Circuits and Systems Magazine*, 21(2):97–118, 2021.

[25] Lana Josipović, Andrea Guerrieri, and Paolo Ienne. From c/c++ code to high-performance dataflow circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2021.

[26] Ioannis Karageorgos, Karthik Sriram, Ján Veselý, Michael Wu, Marc Powell, David A. Borton, Rajit Manohar, and Abhishek Bhattacharjee. Hardware-software co-design for brain-computer interfaces. In *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Valencia, Spain, May 30 - June 3, 2020*, pages 391–404. IEEE, 2020.

[27] Dirk Koch, Frank Hannig, and Daniel Ziener, editors. *FPGAs for Software Programmers*. Springer, 2016.

[28] Rui Li, Lincoln Berkley, Yihang Yang, and Rajit Manohar. Fluid: An asynchronous high-level synthesis tool for complex program structures. In *27th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2021)*, 2021.

[29] Carver Mead and Lynn Conway. *Introduction to VLSI systems*. Addison-Wesley, 1980.

[30] Razvan Nane, Vlad Mihai Sima, Bryan Olivier, Roel Meeuws, Yana Yankova, and Koen Bertels. DWARV 2.0: A cosy-based c-to-vhdl hardware compiler. In Dirk Koch, Satnam Singh, and Jim Tørresen, editors, *22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, August 29-31, 2012*, pages 619–622. IEEE, 2012.

[31] P.G. Paulin and J.P. Knight. Force-directed scheduling for the behavioral synthesis of asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661–679, 1989.

[32] Christian Pilato and Fabrizio Ferrandi. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In *23rd International Conference on Field programmable Logic and Applications, FPL 2013, Porto, Portugal, September 2-4, 2013*, pages 1–4. IEEE, 2013.

[33] A. Putnam et al. Chimps: A c-level compilation flow for hybrid cpu-fpga architectures. In *2008 International Conference on Field Programmable Logic and Applications*.

[34] Parthasarathy Ranganathan, Daniel Stodolsky, Jeff Calow, Jeremy Dorfman, Marisabel Guevara, Clinton Wills Smullen IV, Aki Kuusela, Raghu Balasubramanian, Sandeep Bhatia, Prakash Chauhan, Anna Cheung, In Suk Chong, Niranjani Dasharathi, Jia Feng, Brian Fosco, Samuel Foss, Ben Gelb, Sara J. Gwin, Yoshiaki Hase, Da-ke He, C. Richard Ho, Roy W. Huffman Jr., Elisha Indupalli, Indira Jayaram, Poonacha Kongetira, Cho Mon Kyaw, Aaron Laursen, Yuan Li, Fong Lou, Kyle A. Lucke, J. P. Maaninen, Ramon Macias, Maire Mahony, David Alexander Munday, Srikanth Muroor, Narayana Penukonda, Eric Perkins-Argueta, Devin Persaud, Alex Ramírez, Ville-Mikko Rautio, Yolanda Ripley, Amir Salek, Sathish Sekar, Sergey N. Sokolov, Rob Springer, Don Stark, Mercedes Tan, Mark S. Wachsler, Andrew C. Walton, David A. Wickeraad, Alvin Wijaya, and Hon Kwan Wu. Warehouse-scale video acceleration: co-design and deployment in the wild. In Tim Sherwood, Emery Berger, and Christos Kozyrakis, editors, *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, pages 600–615. ACM, 2021.

[35] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 1989.

[36] John Teifel and Rajit Manohar. Static tokens: Using dataflow to automate concurrent pipeline synthesis. In *10th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2004), 19-23 April 2004, Crete, Greece*, pages 17–27. IEEE Computer Society, 2004.

[37] Richard Townsend, Martha A. Kim, and Stephen A. Edwards. From functional programs to pipelined dataflow circuits. In *Proceedings of the 26th International Conference on Compiler Construction*, CC 2017, page 76–86, New York, NY, USA, 2017. Association for Computing Machinery.

[38] G. Venkataramani, M. Budiu, T. Chelcea, and S. Goldstein. C to asynchronous dataflow circuits: An end-to-end toolflow. In *IWLS*, 2004.

[39] C. G. Wong and A. J. Martin. High-level synthesis of asynchronous systems by data-driven decomposition. In *DAC 2003*.

SCONE: A Logic Locking Technique Utilizing SMT Solver and Circuit Encoding Scheme for Efficient Hardware IP Protection

Zhaokun Han[†], Daniel Xing[‡], Kostas Amberiadis[*], Ankur Srivastava[‡], and Jeyavijayan (JV) Rajendran[†]
[†]Texas A&M University, [‡]University of Maryland, [*]National Institute of Standards and Technology (NIST)
[†]{hzhk0618,jeyavijayan}@tamu.edu, [‡]{dxing97,ankurs}@umd.edu, [*]kostas.amberiadis@nist.gov

***Abstract*—Multiple intellectual property (IP) protections have emerged to defeat security threats in integrated circuit (IC) supply chain. Among these, logic locking is regarded as a promising IP protection for its security. A state-of-the-art work uses stripped-functionality logic locking (SFLL) technique with protected input patterns (PIPs) satisfying the distance of at least 2 (Dist2) property, or D2PIPs, for ensuring resilience against both input-output (I/O)-based and structural attacks. However, this approach has research challenges in scalability, flexibility, and security, as stated and discussed in our paper. Our paper solves these challenges by (i) utilizing a satisfiability modulo theories (SMT) solver and (ii) developing a secure circuit encoding scheme. *SCONE*, our secure logic locking technique, combines the two methods and meets all three challenges simultaneously. Our results show that *SCONE* improves scalability 350× on the IBEX processor (16K gates) and remains resilient against five I/O or structural attacks.**

***Index Terms*—logic locking, encoding scheme, SMT solver.**

I. Introduction

Due to the high cost of integrated circuit (IC) manufacturing, many IC design companies are now fabless and outsource manufacturing, packaging, and testing to third parties. As such, the IC supply chain is highly globalized, with many parties involved in the design and manufacturing process. However, outsourcing leaves ICs vulnerable to supply chain threats, such as intellectual property (IP) piracy, overproduction, counterfeiting, reverse engineering, and Trojan insertion [1]. Many IP protections against these threats have been proposed [2–4]. One promising approach is logic locking, which modifies a design with a key mechanism such that the locked design functions as designed only if given a correct key [2].

One of the most prominent logic locking techniques is stripped-functionality logic locking (SFLL) [5]. SFLL is provably secure against input-output (I/O)-based attacks [6] and practically secure against some structural attacks [7, 8] but vulnerable to others [9, 10]. While Han *et al.* [10] states that SFLL-protected designs with protected input patterns (PIPs) satisfying the distance of at least 2 (Dist2) property, or D2PIPs, can gain structural resilience, such an approach is still impractical due to challenges in scalability, flexibility, and security.

The contributions of this work are as follows:

- We outline and discuss three challenges (i.e., scalability, flexibility, and security) in the state-of-the-art work [10].
- To solve the scalability challenge, we propose a novel approach using a satisfiability modulo theories (SMT) solver.
- The paper develops a novel secure circuit encoding scheme, enhancing both flexibility and security.
- By combining the two novel methods (using an SMT solver and developing a secure encoding), the proposed logic locking technique, *SCONE*, solves all three challenges simultaneously and keeps resilient against all five I/O and structural attacks [6–10].
- Based on the designer's capability and preference, this paper introduces two implementations of *SCONE*, including a discussion of their performance comparison.

In the remainder of the paper, Section II introduces the background, Section III introduces the proposed logic locking technique, *SCONE*, Section IV presents results, Section V provides the discussions, and Section VI concludes the contributions.

II. Background

A. Potential Attacks on SFLL

Potential attacks on SFLL with D2PIPs [10] can be categorized into (i) I/O attacks and (ii) structural attacks.

I/O attacks. I/O attacks challenge the design's IP by querying input patterns and observing correct outputs from an activated chip purchased from the market (a.k.a. oracle). The attacker determines the queries by analyzing the locked netlist. The most powerful I/O attack is the *Boolean satisfiability (SAT) attack [6]*, as it breaks all logic locking techniques prior to the SAT attack. A SAT-capable attacker has a locked netlist and can query an oracle. The SAT attack eliminates incorrect keys iteratively by querying a distinguishing input pattern (DIP) till discovering the correct key.

Structural attacks. The locked netlist may contain clues to the correct key, which structural attacks exploit by analyzing the structure of the locked netlist. There are various structural attacks. The *AND-tree removal (ATR) attack [7]* identifies the AND-tree with a point function, which contains exactly one input pattern resulting in output 1, in the netlist and removes that tree to recover the functionality. The *signal probability skew (SPS) attack [8]* locates the wires with point functions by analyzing the probability of the signal to be 0/1 in the locked netlist and bypasses those wires to recover the functionality. The *function-analysis attack on logic locking (FALL) [9]* relies on the *unateness* property to locate key units and bypass them to retrieve the functionality. The *sparse prime implicant (SPI) attack [10]* analyzes the locked circuit's prime implicant (PI) table[1] to infer the PIP information and unlock the design.

This work is supported in part by NSF 1953285 and CSME 210205 grants.

DISCLAIMER: The opinions, recommendations, findings, and conclusions in this article do not necessarily reflect the views or policies of NIST or the United States Government.

[1]A PI table can be considered as a sum-of-products, where each PI represents a product. For all PI, all input bit values are from $\{0, 1, \text{-}\}$. The symbol "-" denotes a don't-care bit, meaning its value does not affect whether the PI belongs to the ON-set/OFF-set.

B. Previous Approach: Using SFLL with D2PIPs

Using SFLL with D2PIPs [10] to protect the design provides provable security against both I/O and structural attacks.

An SFLL-protected circuit has two main components: a stripped-functionality circuit (FSC) and a restore unit [5]. The FSC obfuscates protected input patterns (PIPs)—the FSC's output differs from the original circuit's output only when the input pattern is a PIP. With the correct key, the restore unit corrects the FSC's flipped output. By XORing the FSC and the restore unit, the designer generates the SFLL-protected circuit.

An attacker can break a SFLL-protected design by finding the PIPs. However, for the SAT attacker, the probability of finding a PIP is exponentially small to each PIP's length. Thus, SFLL is provably secure against the SAT attack [5]. There are also multiple variants of SFLL [5, 11–13], each targeting specific objectives such as better security, lower overhead, or a higher error rate. However, all these SFLL techniques are broken by one of the aforementioned attacks, especially the SPI attack [10]. Only the state-of-the-art work [10] survives for its sufficient security against all aforementioned attacks.

SFLL is practically secure against some structural attacks (i.e., ATR [7] and SPS [8]) but vulnerable to other structural attacks (i.e., FALL [9] and SPI [10]). As stated in Han *et al.* [10], for most randomly selected PIPs, the FSC's netlist can leak PIP information, leading to structural vulnerabilities; thus, the defender needs to select specific PIPs to safeguard the design from structural information leakage. Han *et al.* [10] proposes that, from the PI table of the original circuit,[2] the defender can select PIs that keep at least distance 2 away from all other PIs in the PI table, as indicated in

$$\{D2PIP_{PI}\} = \{pi^* \,|\, \forall pi \neq pi^* \in F_{orig}^{ON},\, Dist(pi^*, pi) \geq 2\}. \quad (1)$$

This property is referred to as the "Dist2 property," and PIs meeting this criterion are "D2PIPs." By choosing the D2PIPs as the PIPs, the FSC can conceal the PIPs information [10].

The previous work [10] protects the design with D2PIPs with the flow of (i) converting the netlist into a PI table for each output, (ii) analyzing each PI table to collect D2PIPs, and (iii) applying SFLL with the collected D2PIPs to generate the SFLL-protected circuit. The design protected by SFLL with D2PIPs is secure against both I/O and structural attacks. However, research challenges exist in this approach, and we will demonstrate the challenges in Section III.

III. Proposed Logic Locking Technique, *SCONE*

This section first outlines the challenges of scalability, flexibility, and security in the previous approach [10] (see Section III-A). To address all challenges, the paper introduces two methods: (i) utilizing an SMT solver to directly extract D2PIPs from the netlist, thereby achieving scalability (see Section III-B), and (ii) developing a circuit encoding scheme to introduce more secure PIPs, ensuring both flexibility and security (see Section III-C). Lastly, we propose a logic locking technique, *SCONE*, that safeguards the design against I/O and structural attacks while solving all three challenges simultaneously (see Section III-D).

[2]For simplicity, we focus on analyzing the ON-set of the original circuit, as well as the PIs and D2PIPs. A similar deduction can be made when considering the OFF-set case.

A. Research Challenges in Previous [10]

Scalability. To collect D2PIPs, the previous approach [10] involves converting the netlist into the PI table for each output. However, this circuit format conversion is a nondeterministic polynomial (NP) -hard process that scales exponentially to the circuit input size. For medium- and large-scale designs, this conversion is time-consuming and may not terminate within a reasonable time limit, resulting in a lack of scalability.

Flexibility. The high complexity of circuit format conversion often hinders the success of the PI table extraction within a reasonable time, and thereby, collected D2PIPs are rare. Moreover, for most practical designs, even assuming the success of every PI table extraction with an impractically huge time limit, D2PIPs are still rare.[3] Consequently, the rareness of D2PIPs restricts the defender's flexibility in selecting D2PIPs.

Security. The D2PIP's length is often too small to protect the design from brute-force attacks due to the rareness of D2PIPs.

B. Fast D2PIPs Extraction Using an SMT Solver

We propose using the SMT solver to directly extract D2PIPs from the netlist, bypassing the process of generating a PI table.

According to Equation (1), for a D2PIP, all its distant-1 minterms[4] in the Boolean space must be OFF-set. Otherwise, there exists a PI covering an ON-set minterm distant-1 away from the D2PIP, resulting in a contradiction. Thus,

$$\{D2PIP_{PI}\} = \{pi^* \in F_{orig}^{ON} \,|\, \forall m,\, Dist(m, pi^*) = 1 \Rightarrow f(m) = 0\}. \quad (2)$$

In Equation (2), pi^* is an ON-set D2PIP, "$\forall m, Dist(m, pi^*) = 1$" denotes all minterm m's distant-1 to pi^*, and "$\Rightarrow f(m) = 0$" represents the implies/implication of the OFF-set minterm m. However, Equation (2)'s condition still costs a high-execution effort for the SMT solver to search for D2PIPs. Thus, we narrow down to search for minterm D2PIPs only for the highest brute-force resilience, as opposed to PI D2PIPs with don't cares and thereby less resilience. For minterm D2PIPs,

$$\{D2PIP_m\} = \{m^* \in F_{orig}^{ON} \,|\, \forall m,\, Dist(m^*, m) = 1 \Rightarrow f(m) = 0\}. \quad (3)$$

Using the SMT solver, we address the scalability challenge of searching for D2PIPs. Next, we demonstrate how the SMT solver searches for minterm D2PIPs based on Equation (3).

For a minterm D2PIP $m^* \in \{0,1\}^n$ (n denotes the input size), we need the SMT solver to return the solution of each minterm D2PIP iteratively. For every minterm m distant-1 to m^*, there are n candidate m's, each flipping exactly one bit of m^*. Additionally, with the implies/implication "$\Rightarrow f(m) = 0$", the SMT solver applies the circuit's Boolean logic of the output $f(\cdot)$ with ensuring that minterm m is OFF-set. In this way, the SMT solver detects a minterm D2PIP m^*. Iteratively, we collect other minterm D2PIPs. Theoretically, we can collect all minterm D2PIPs. However, if the circuit contains numerous minterm D2PIPs that cannot be fully collected within a reasonable time, we choose to gather partial minterm D2PIPs since we do not require all D2PIPs for protection.

[3]The rareness of D2PIPs is an inherent attribute for most practical designs. The designer usually applies cost-effective encoding (e.g., binary, Gray, and one-hot [14]) to produce compact and high-performance designs, which could result in this rareness [10].

[4]A minterm is a product where all input bit values are from $\{0, 1\}$ (no don't care bit).

Although we can efficiently and successfully analyze all outputs by using a SMT solver, there are still very few minterm D2PIPs due to the rareness of D2PIPs and the restrictive condition of minterm D2PIPs. To address the challenges of flexibility and security, the next subsection proposes a novel method to introduce more secure PIPs with sufficient lengths.

C. Secure Circuit Encoding Scheme

We propose a circuit encoding scheme that (i) offers flexible choices of secure PIPs, and (ii) keeps resilient against I/O and structural attacks and gains security against brute-force attacks.

Assume the original circuit, $\mathcal{C}_{orig}$, has n inputs $(in_1, in_2, \cdots, in_n)$. To introduce secure minterm PIP(s), the defender firstly selects ON-set minterm PIP(s) with the length of n. This selection provides the defender flexibility in choosing PIP(s). Assume that one selected PIP is $PIP^{before} = a_1 a_2 \cdots a_n$, where $a_i \in \{0, 1\}$ and $i \in [1, n] \cap \mathbb{N}$, we need to expand inputs in the encoded circuit, $\mathcal{C}_{encoded}$. To transform this selected PIP PIP^{before} into a PIP PIP^{after} with a longer length of $n + m$ ($m \in \mathbb{N}^+$), $\mathcal{C}_{encoded}$ requires more input bits. Assume $\mathcal{C}_{encoded}$'s inputs are $(in_1^{encoded}, in_2^{encoded}, \cdots, in_{n+m}^{encoded})$, where $in_i^{encoded} = in_i$ for $i \in [1, n] \cap \mathbb{N}$. With m extra input bits, the Boolean space size increases exponentially, thereby enhancing brute-force resilience. Moreover, the choice of m provides the defender flexibility in determining the PIP size. When $i \in [n+1, n+m] \cap \mathbb{N}$, $in_i^{encoded}$'s encoding is determined by a random selection of $(in_{c_1^{i-n}}, in_{c_2^{i-n}}, \cdots, in_{c_w^{i-n}})$, where $c_1^{i-n} \leq c_2^{i-n} \leq \cdots \leq c_w^{i-n} \in \{1, 2, \cdots, n\}$ and $w \in [2, n-1] \cap \mathbb{N}$. In this secure circuit encoding scheme, we define $in_i^{encoded} = \overline{in_{c_1^{i-n}} \oplus in_{c_2^{i-n}} \oplus \cdots \oplus in_{c_w^{i-n}}}$ when $i \in [n+1, n+m] \cap \mathbb{N}$. Therefore,

$$in_i^{encoded} = \begin{cases} in_i, & \text{if } i \in [1, n] \cap \mathbb{N} \\ \overline{in_{c_1^{i-n}} \oplus \cdots \oplus in_{c_w^{i-n}}}, & \text{if } i \in [n+1, n+m] \cap \mathbb{N}. \end{cases} \tag{4}$$

After encoding, $PIP^{after} = b_1 b_2 \cdots b_{n+m}$ satisfies

$$b_i = \begin{cases} a_i, & \text{if } i \in [1, n] \cap \mathbb{N} \\ \overline{a_{c_1^{i-n}} \oplus a_{c_2^{i-n}} \oplus \cdots \oplus a_{c_w^{i-n}}}, & \text{if } i \in [n+1, n+m] \cap \mathbb{N}. \end{cases} \tag{5}$$

Theoretically, there are total $\binom{n}{w}$ combinations for choosing $(c_1^j, c_2^j, \cdots, c_w^j)$; we set $w = \lfloor \frac{n}{2} \rfloor$ for providing the highest difficulty to the attacker by achieving the maximum of $\binom{n}{w}$. Moreover, to ensure that PIP^{after} satisfies the Dist2 property, we choose $m \geq 2$ for maintaining resilience against structural attacks. In the next subsection, we describe how we apply this encoding to protect the design by utilizing the SMT solver.

D. Logic Locking Technique, SCONE

We aim to protect the design while solving three challenges (i.e., scalability, flexibility, and security). This is accomplished by combining the approach of using the SMT solver for fast D2PIPs extraction with the secure encoding that introduces flexible and secure PIPs in the expanded Boolean space.

We now discuss the protection flow of *SCONE*. Firstly, we prefer to choose minterm D2PIPs if they are available. By utilizing the SMT solver, we can efficiently and effectively extract minterm D2PIPs. If no minterm D2PIP exists, we then apply the encoding to introduce flexible and secure PIPs and protect the design. A *SCONE*-protected design after encoding consists of three main components: an encoding stage (ES), an FSC of $\mathcal{C}_{encoded}$, and a restore unit. The relationship between inputs and outputs of the ES (IN_{orig} and $IN_{encoded}$) satisfies Equation (4). When with multiple PIP^{after}'s, $\{PIP_1^{after}, PIP_2^{after}, \cdots, PIP_p^{after}\}$ ($p \geq 2$), the correct key is the concatenate, $key_c = \prod_{i=1}^{p} PIP_i^{after}$. We have two implementations based on whether the ES is implemented in hardware (HW):

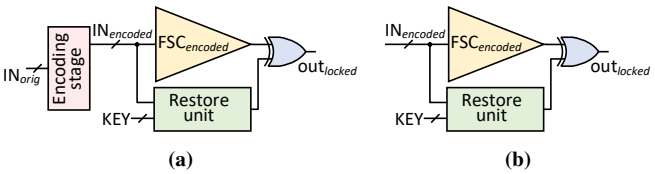


Fig. 1: The structure diagram of the *SCONE*-protected design in the implementation of (a) with ES in HW or (b) without ES in HW.

Implementation with ES in HW (see Fig. 1(a)): If the designer does not apply the encoding during the design stage and maintains the original circuit's functionality, then the circuit keeps the original inputs IN_{orig} with a size of n. In this case, the defender needs to implement the configurable ES in HW to obfuscate the secure encoding. Notably, with the same input size of n, the resilience against brute-force attacks remains unchanged because the Boolean space keeps the same size of 2^n. In this implementation, the threat model allows the attacker to access (i) an oracle of the original design and (ii) the logic-locked netlist as shown in Fig. 1(a). The attacker's goal is to recover the original circuit's functionality.

Implementation without ES in HW (see Fig. 1(b)): The designer applies the secure encoding during the design stage, and the inputs of the produced design are $IN_{encoded}$, with a size of $n + m$. The attacker does not know how the defender determines the encoding as described in Equation (4). This implementation expands the input bits from n to $n+m$, thereby enhancing security against brute-force attacks (and other I/O attacks including the SAT attack) by significantly increasing the size of the Boolean space. Different from the previous threat model, in this implementation, the attacker is allowed to access (i) the oracle of the encoded circuit and (ii) the logic-locked netlist as shown in Fig. 1(b). The attacker aims to retrieve the encoded circuit's functionality.

IV. Results

This section first discusses the experimental setup (Section IV-A). Then, we compare the distribution of D2PIPs and the execution time for extracting D2PIPs between the previous approach [10] and *SCONE* to highlight the flexibility of *SCONE* in Section IV-B. Next, we demonstrate the scalability by showing the execution time required by *SCONE* in Section IV-C. Following this, we conduct a security analysis with various attacks results on *SCONE*-protected circuits in Section IV-D. Finally, we present an overhead analysis in terms of power, performance, and area (PPA) in Section IV-E.

A. Experimental Setup

We perform experiments on a 32-core Intel Xeon processor at 2.6 GHz with 512 GB RAM. We use *Berkeley ABC* tool [15] for circuit format conversion, *Synopsys Design Compiler* [16]

for (re-)synthesis, and *Microsoft z3* as the SMT solver.[5] Our test circuits consist of four ISCAS'85 circuits [17], nine ITC'99 circuits [18], and three other circuits (i.e., *Stanford MIPS* [19], *RISC-V IBEX* [20], and *Common Evaluation Platform (CEP) Global Positioning System (GPS)* [21]).

B. Collecting D2PIPs

Section III claims that (i) using the proposed approach utilizing the SMT solver accelerates extracting D2PIPs and (ii) using the circuit encoding scheme introduces flexible choices of minterm D2PIPs, compared to the previous approach [10]. Table I shows this comparison. Columns 3 & 6 present the success rate of extracting D2PIPs from the design, columns 4 & 7 reveal the maximum of D2PIP sizes, and columns 5 & 8 show the execution time to finish the D2PIPs extraction. Firstly, we can tell that the previous approach cannot ensure the success rate of D2PIPs extraction, while *SCONE* ensures the success rate of 100%. Next, *SCONE* is much more flexible than the one for the previous approach [10] for its enlarged maximum PIP size. "N.A." denotes that the previous approach [10] fails to extract any D2PIP. Additionally, the maximum PIP size that *SCONE* can achieve is significantly higher than that of the previous approach [10], thereby enhancing security against brute-force attacks. Lastly, *SCONE* provides a much quicker analysis time than the previous approach [10]. "T.O." denotes that the previous approach [10] cannot finish the D2PIPs extraction within 72 hours. Therefore, *SCONE* achieves more flexible PIP choices much faster while ensuring higher resilience against brute-force attacks, compared to the previous approach [10].

C. Execution Time of SCONE

Table II shows, for each circuit, the execution time required by *SCONE* when protecting the design. The chosen PIP sizes vary for different circuits. Here, we consider both implementations: one with ES in HW and one without ES in HW. From columns 4 & 5, we observe that the execution time required by the implementation without ES in HW is significantly less than that with ES in HW. Compared to the implementation without ES in HW, the implementation with ES in HW requires additional execution time for building the configurable ES in HW, increasing the execution time. For the implementation with ES in HW, the defender first builds the configurable ES in a register-transfer level (RTL) Verilog file. Then, the defender uses *Synopsys Design Compiler* [16] to synthesize the RTL ES into its gate-level netlist. Finally, the defender combines the SFLL-protected $\mathcal{C}_{encoded}$ with the netlist of the configurable ES. As a result, building ES in HW requires significant extra time. Notably, even for large-scale designs such as IBEX and GPS, the execution time required by *SCONE* for the implementation without ES in HW is less than 80 seconds, demonstrating the scalability of *SCONE*.

We also include an analysis of the same design with varying PIP sizes. Fig. 2 shows how the protection execution time changes with different PIP sizes for the b14 design when using *SCONE*. From this figure, we observe that, for both implementations, the one without ES in HW always requires less time than the one with ES in HW. Additionally, varying the PIP size does not significantly affect the execution time for the implementation without ES in HW, indicating stable execution time with different PIP sizes. Thus, this result also supports the scalability of *SCONE*.

D. Security Analysis

In this subsection, we first have security analysis on *SCONE*; then, we showcase the vulnerability or infeasibility of existing SFLL techniques with attack results.

Security Analysis on *SCONE*. To analyze the security of *SCONE*, we present Table III with all attack results (i.e., SAT [6], SPS [8], ATR [7], FALL [9], and SPI [10]) on the *SCONE*-protected circuits, where "T.O." denotes that the SAT attack cannot terminate within 72 hours, and ✗ indicates that

[5]We choose *Microsoft's z3* solver since it is a standard and widely-used SMT solver. Our theory is compatible with all SMT solvers, so other SMT solvers can also be used.

TABLE I: Comparison of D2PIPs extraction between the previous approach [10] and *SCONE*.

Circuit		Previous [10]			*SCONE*		
		Success rate (%)	Max. PIP size	Max. time (sec)	Success rate (%)	Max. PIP size	Max. time (sec)
ISCAS'85	c432	100	N.A.	431.6	100	72	1.9
	c1355	0	N.A.	T.O.	100	82	4.3
	c1908	16.0	N.A.	T.O.	100	66	3.1
	c7552	68.2	4	T.O.	100	388	45.8
ITC'99	b11	100	9	0.4	100	52	2.2
	b12	100	26	0.4	100	74	1.7
	b13	100	11	0.1	100	46	0.4
	b14	53.5	29	T.O.	100	436	89.7
	b15	75.9	7	T.O.	100	612	281.7
	b17	60.4	6	T.O.	100	678	746.6
	b20	40.0	29	T.O.	100	570	305.3
	b21	35.2	29	T.O.	100	576	315.4
	b22	34.8	29	T.O.	100	578	322.2
Others	MIPS	80.6	3	T.O.	100	310	46.7
	IBEX	48.0	39	T.O.	100	902	748.2
	GPS	97.9	20	T.O.	100	240	102.9

TABLE II: The execution time of protecting the test circuit required by *SCONE*, including two different implementations.

Circuit		PIP size	Time (sec)	
			With ES in HW	Without ES in HW
ISCAS'85	c432	38	29.8	9.8
	c1355	43	36.3	12.8
	c1908	35	29.5	11.9
	c7552	126	314.3	26.8
ITC'99	b11	28	24.0	10.2
	b12	39	30.0	9.9
	b13	25	20.4	8.4
	b14	102	211.2	29.9
	b15	129	229.7	40.5
	b17	117	354.7	130.4
	b20	103	202.8	54.9
	b21	103	211.2	61.1
	b22	103	194.9	47.5
Others	MIPS	37	16.4	6.5
	IBEX	102	257.9	74.5
	GPS	103	226.3	77.5

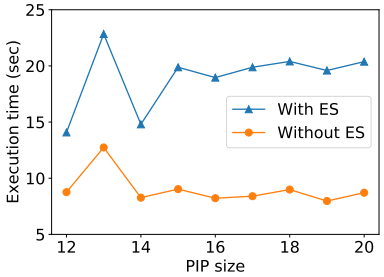


Fig. 2: The execution time of *SCONE* on b14 with different PIP sizes.

TABLE III: Attack results on *SCONE*-protected circuits.

	Circuit	PIP size	With ES in HW					Without ES in HW				
			I/O	Structural				I/O	Structural			
			SAT	SPS	ATR	FALL	SPI	SAT	SPS	ATR	FALL	SPI
ISCAS'85	c432	38	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	c1355	43	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	c1908	35	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	c7552	126	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
ITC'99	b11	28	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	b12	39	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	b13	25	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	b14	102	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	b15	129	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	b17	117	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	b20	103	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	b21	103	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	b22	103	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
Others	MIPS	37	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	IBEX	102	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗
	GPS	103	T.O.	✗	✗	✗	✗	T.O.	✗	✗	✗	✗

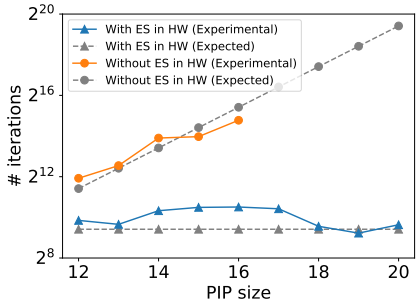
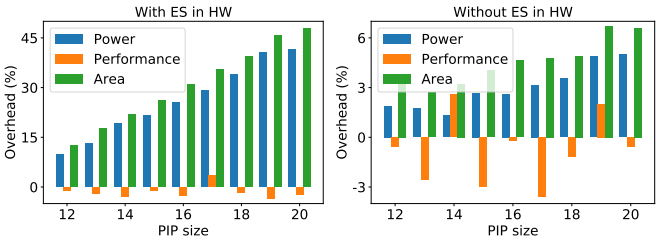


Fig. 3: The number of iterations required by the SAT attack [6] on b14 with different PIP sizes. The y-axis has an exponential scale.

the structural attack fails to break the design. All *SCONE*-protected circuits are resilient against any tested attack. Additionally, the PIP sizes chosen in Table III for *SCONE* are always greater than the maximum PIP sizes achievable by the previous approach [10]. For example, with GPS, the brute-force search space size is enlarged from 2^{39} (for the previous approach [10]) to 2^{103} (for *SCONE*). This indicates the much more enhanced resilience against brute-force attacks.

The SAT attack times out after 72 hours on all test circuits listed in Table III due to the large PIP sizes. To better understand the security of *SCONE* against the SAT attack, we present the SAT attack results on b14 with relatively small and varied PIP sizes in Fig. 3. The blue triangle marker denotes the experimental number of SAT iterations for the implementation with ES in HW, and the orange dot marker denotes the experimental number of iterations for the implementation without ES in HW. The grey triangle/dot marker denotes the expected number of iterations for the implementation with/without ES in HW. We can observe that, when changing the PIP size, the number of iterations is stable in a small range for the implementation with ES in HW, close to the expected number (the grey triangle markers in Fig. 3). This is because the SAT resilience is limited by the input size of the original circuit. Differently, when applying ES during the design stage (referring to the implementation without ES in HW), the number of iterations increases exponentially to the PIP size when the PIP size is in range of $[12, 16] \cap \mathbb{N}$, as shown in Fig. 3, where the y-axis is on a log_2 scale. This tendency is close to the expected number of iterations (the grey dot markers in Fig. 3). When the PIP size is ≥ 17, the SAT attack times out after 72 hours. Compared to the implementation with ES in HW, the one without ES in HW provides more I/O resilience and requires an exponentially large number of SAT iterations to the PIP size according to the curve's tendency.



(a) (b)

Fig. 4: PPA overhead analysis on b14 with different PIP sizes.

Security Analysis on Existing SFLL Techniques. We aim to showcase the insufficient resilience or infeasibility of the existing SFLL techniques, including SFLL-HD [5], SFLL-flex [5], and the state-of-the-art approach [10]. Table IV presents the attack results on these existing SFLL techniques. For both SFLL-HD and SFLL-flex, all locked circuits are vulnerable to the SPI attack [10]. For the state-of-the-art approach, using SFLL with D2PIPs as described in Han *et al.* [10], the test circuit is either not able to be protected due to the lack of D2PIP, or the PIP size is too small to provide brute-force resilience thereby resulting in insufficient SAT resilience. Thus, not only the state-of-the-art approach [10] but also other SFLL techniques [5], *SCONE* not only addresses the limitations in scalability, flexibility, and security, but also does not reduce resilience against various attacks [6–10]. Thus, *SCONE* outperforms the existing SFLL techniques.

E. Overhead Analysis

Table V presents the PPA overhead results of two implementations. We can observe that without ES in HW incurs a lower overhead than the other one, particularly in power and area. Notably, for large-scale designs (e.g., IBEX and GPS), the PPA overheads are $< 10\%$ in area and power and with a negative performance overhead (improved performance). Thus, the implementation without ES in HW provides a more tolerable and acceptable PPA overhead.

To understand how the PPA overhead varies when changing the PIP size, we provide the PPA overhead analysis on b14 with different PIP sizes, as shown in Fig. 4. For both implementations, the power and area overheads go approximately linear to the PIP size. The negative performance overhead indicates the improved performance using *SCONE*, which is also common for other SFLL techniques [5, 11]. Moreover, the implementation without ES in HW always has lower power and area overheads for avoiding the high cost in the configurable ES circuit.

V. Discussion

This section discusses (i) the enhanced performance of *SCONE* in scalability, flexibility, and security compared to the previous approach [10], (ii) the comparison between two implementations, and (iii) prior related research works.

TABLE IV: Attack results on the existing SFLL techniques, including SFLL-HD [5], SFLL-flex [5], and the state-of-the-art approach [10].

Circuit		SFLL-HD [5]						SFLL-flex [5]						Previous [10]					
		PIP size	I/O	Structural				PIP size	I/O	Structural				PIP size	I/O	Structural			
			SAT	SPS	ATR	FALL	SPI		SAT	SPS	ATR	FALL	SPI		SAT	SPS	ATR	FALL	SPI
ISCAS'85	c432	36	311	✗	✗	✗	✓	36	115	✗	✗	✗	✓	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
	c1355	41	T.O.	✗	✗	✗	✓	41	105	✗	✗	✗	✓	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
	c1908	33	165	✗	✗	✓	✓	33	217	✗	✗	✗	✓	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
	c7552	207	7,470	✗	✗	✓	✓	207	T.O.	✗	✗	✗	✓	4	12	✗	✗	✗	✗
ITC'99	b11	38	T.O.	✗	✗	✗	✓	38	62	✗	✗	✗	✓	9	434	✗	✗	✗	✗
	b12	126	T.O.	✗	✗	✗	✓	126	619	✗	✗	✗	✓	26	T.O.	✗	✗	✗	✗
	b13	63	T.O.	✗	✗	✗	✓	63	T.O.	✗	✗	✗	✓	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
	b14	277	7,472	✗	✗	✗	✓	277	T.O.	✗	✗	✗	✓	17	T.O.	✗	✗	✗	✗
	b15	485	2,537	✗	✗	✗	✓	485	T.O.	✗	✗	✗	✓	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
	b17	1,452	T.O.	✗	✗	✗	✓	1,452	T.O.	✗	✗	✗	✓	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
	b20	522	2,281	✗	✗	✗	✓	522	T.O.	✗	✗	✗	✓	17	T.O.	✗	✗	✗	✗
	b21	522	T.O.	✗	✗	✗	✓	522	3,794	✗	✗	✗	✓	17	T.O.	✗	✗	✗	✗
	b22	767	T.O.	✗	✗	✗	✓	767	T.O.	✗	✗	✗	✓	17	T.O.	✗	✗	✗	✗
Others	MIPS	466	T.O.	✗	✗	✗	✓	466	350	✗	✗	✗	✓	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
	IBEX	1,386	T.O.	✗	✗	✗	✓	1,386	T.O.	✗	✗	✗	✓	5	29	✗	✗	✗	✗
	GPS	10,009	T.O.	✗	✗	✗	✓	10,009	T.O.	✗	✗	✗	✓	20	T.O.	✗	✗	✗	✗

TABLE V: Overhead analysis for *SCONE*'s two implementations.

Circuit		# gates in $\mathcal{C}_{orig}$	Overhead (%)					
			With ES in HW			Without ES in HW		
			Power	Performance	Area	Power	Performance	Area
ISCAS'85	c432	154	2624.7	11.6	2720.6	205.5	3.5	216.8
	c1355	191	882.2	13.9	1634.3	73.6	14.4	115.3
	c1908	218	814.0	-2.2	1213.7	81.0	-6.1	110.5
	c7552	1,145	1914.4	-7.7	2935.5	74.5	-2.8	89.5
ITC'99	b11	445	523.6	7.1	506.6	53.0	-1.8	51.8
	b12	986	572.6	-1.6	411.5	53.5	-2.2	33.1
	b13	253	864.9	8.0	691.3	99.0	-2.0	77.8
	b14	3,534	558.3	0.6	724.9	18.0	2.6	21.5
	b15	6,006	616.6	-4.5	578.1	16.2	-6.8	14.9
	b17	24,498	133.1	-4.2	130.5	3.6	-4.9	3.1
	b20	7,982	203.1	2.5	311.9	7.4	1.7	8.5
	b21	8,117	186.0	-2.1	302.6	7.0	-4.3	8.4
	b22	12,143	136.1	-1.5	204.7	3.3	-3.1	5.5
Others	MIPS	3,046	147.0	-8.4	119.2	11.6	-5.2	8.7
	IBEX	13,195	200.5	1.3	189.9	7.5	-4.2	7.1
	GPS	52,464	41.7	7.0	46.4	1.7	-3.3	2.3

A. Improvements in Three Perspectives

The experimental results and the related analysis indicate that *SCONE* overcomes the challenges in scalability, flexibility, and security. Table VI summarizes the comparison by listing the limitations in the previous approach [10] and the improvements that *SCONE* achieves. The summary table demonstrates the effectiveness of *SCONE* in overcoming three challenges.

TABLE VI: Comparison of the previous approach [10] and *SCONE*.

Perspective	Limitation in Previous [10]	Improvement in *SCONE*
Scalability	Exhausting search for D2PIPs and cannot guarantee success.	Fast D2PIPs extraction with a success rate of 100%.
Flexibility	D2PIPs are rare or do not exist.	Flexible choice of D2PIPs.
Security	Resilient against I/O and structural attacks but vulnerable to brute-force attacks.	Resilient against brute-force attacks while keeping resilient against I/O and structural attacks.

B. Two Implementations

There are two implementations of *SCONE* proposed in this paper, with and without ES in HW. Section IV demonstrates the strengths of the one without ES in HW from multiple perspectives: less execution time required by *SCONE*, higher resilience against SAT and brute-force attacks, and lower overheads (especially in power and area). Thus, if the designer is capable of applying the encoding during the design stage, we recommend choosing the implementation without ES in HW. The implementation without ES in HW (i) is more scalable in terms of execution time, (ii) provides higher security against I/O attacks, and (iii) holds a lower PPA overhead.

C. Prior Works on Secure Encoding Schemes

There are some prior and related works on secure encoding. *Active Metering [22]* is the first work that proposes a secure encoding scheme for hardware metering. The protected design has additional flip-flops (FFs) and obfuscates the correct initial state. However, this technique is vulnerable [23]. *TriLock [24]* introduces a state encoding scheme for sequential logic locking for security purposes by encoding and decoding the inputs and outputs of FFs, and protects the design using the point-function method. One needs to be aware of the usage of the point function due to its potential structural vulnerabilities [7–10]. *STATION [25]* proposes *DisJoint* encoding, a prior secure encoding scheme for SFLL. The *DisJoint*-encoded circuit has more D2PIP choices, providing the defender flexibility. However, *DisJoint* encoding targets FSM obfuscation only and requires the state-transition table (SST) of the original design. Converting a netlist to its SST is a time-consuming/exhausting process, so the scalability challenge exists in *STATION* as well.

D. Machine Learning-based Attacks

Other prior works have examined machine learning (ML)-based approaches to attack logic locking. They can be broadly categorized by their methods: those based on more conventional machine learning methods that work on fixed-dimension extracted features from locked circuits (e.g. *SAIL* [26], *SnapShot* [27], *SCOPE* [28]) and those that utilize graph neural networks which work directly on the graph structure of the locked netlist (e.g. *OMLA* [29], *INSIGHT* [30]). All of the previous works consider only key-gate insertion-based logic locking methods and do not challenge SFLL techniques. Since *SCONE* protects the design's IP by relying on the mechanism of SFLL, *SCONE* is resilient against existing ML-based attacks in logic locking.

VI. Conclusion

This paper proposes the secure logic technique, *SCONE*, addressing all challenges: scalability, flexibility, and security. Depending on the capability, the designer can choose the implementation either with or without ES in HW. We recommend the implementation without ES in HW for a faster protection process, higher security level, and a lower PPA overhead.

REFERENCES

[1] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[2] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," *IEEE/ACM Design, Automation and Test in Europe*, pp. 1069–1074, 2008.

[3] R. P. Cocchi, L. W. Chow, J. P. Baukus, and B. J. Wang, "Method and Apparatus for Camouflaging a Standard Cell based Integrated Circuit with Micro Circuits and Post Processing," *US Patent no. 20120139582*, 2012. [Online]. Available: https://patents.google.com/patent/US20120139582/en

[4] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation," *USENIX Security Symposium*, pp. 495–510, 2013. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/imeson

[5] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-Secure Logic Locking: From Theory To Practice," *ACM SIGSAC Conference on Computer and Communications Security*, p. 1601–1618, 2017. [Online]. Available: https://doi.org/10.1145/3133956.3133985

[6] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 137–143, 2015.

[7] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably Secure Camouflaging Strategy for IC Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1399–1412, 2019.

[8] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security Analysis of Anti-SAT," *IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 342–347, 2017.

[9] D. Sirone and P. Subramanyan, "Functional Analysis Attacks on Logic Locking," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2514–2527, 2020.

[10] Z. Han, M. Yasin, and J. Rajendran, "Does logic locking work with EDA tools?" *USENIX Security Symposium*, pp. 1055–1072, 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/han-zhaokun

[11] A. Sengupta, M. Nabeel, N. Limaye, M. Ashraf, and O. Sinanoglu, "Truly Stripping Functionality for Logic Locking: A Fault-Based Perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4439–4452, 2020.

[12] B. Tan, R. Karri, N. Limaye, A. Sengupta, O. Sinanoglu, M. M. Rahman, S. Bhunia, D. Duvalsaint, R. D., Blanton, A. Rezaei, Y. Shen, H. Zhou, L. Li, A. Orailoglu, Z. Han, A. Benedetti, L. Brignone, M. Yasin, J. Rajendran, M. Zuzak, A. Srivastava, U. Guin, C. Karfa, K. Basu, V. V. Menon, M. French, P. Song, F. Stellari, G.-J. Nam, P. Gadfort, A. Althoff, J. Tostenrude, S. Fazzari, E. Breckenfeld, and K. Plaks, "Benchmarking at the Frontier of Hardware Security: Lessons from Logic Locking," *arXiv preprint arXiv:2006.06806*, 2020. [Online]. Available: https://arxiv.org/abs/2006.06806

[13] Z. Han, M. Yasin, and J. Rajendran, "Multi-Objective Strategies for Stripped-Functionality Logic Locking," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.

[14] W. Meeus, "Finite State Machine (FSM) encoding in VHDL: binary, one-hot, and others," https://insights.sigasi.com/tech/vhdl-onehot-fsm/, 2020, Last accessed on 09/08/2021.

[15] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," *International Conference on Computer Aided Verification (CAV)*, pp. 24–40, 2010. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-14295-6_5

[16] Synopsys, "Design Compiler NXT," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html, 2024, Last accessed on 04/02/2024.

[17] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[18] S. Davidson, "Characteristics of the ITC'99 Benchmark Circuits," *IEEE International Test Synthesis Workshop*, p. 87, 1999. [Online]. Available: https://www.cerc.utexas.edu/itc99-benchmarks/itsw.bench.pdf

[19] J. Hennessy, N. Jouppi, F. Baskett, and J. Gill, "MIPS: A VLSI Processor Architecture," *Springer VLSI Systems and Computations*, pp. 337–346, 1981. [Online]. Available: https://doi.org/10.1007/978-3-642-68402-9_37

[20] L. Woods, Z. István, and G. Alonso, "Ibex: An Intelligent Storage Engine with Support for Advanced SQL Off-loading," *VLDB Endowment*, vol. 7, no. 11, p. 963–974, 2014. [Online]. Available: https://doi.org/10.14778/2732967.2732972

[21] M. Hicks, P. Fiscarelli, and B. Chetwynd, "Common Evaluation Platform," https://github.com/mit-ll/CEP, 2021, Last accessed on 09/10/2024.

[22] Y. M. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," *USENIX Security Symposium*, vol. 20, pp. 1–20, 2007. [Online]. Available: https://www.usenix.org/conference/16th-usenix-security-symposium/active-hardware-metering-intellectual-property-protection

[23] M. Fyrbiak, S. Wallat, J. Déchelotte, N. Albartus, S. Böcker, R. Tessier, and C. Paar, "On the Difficulty of FSM-based Hardware Obfuscation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 293–330, 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7277

[24] Y. Zhang, Y. Hu, P. Nuzzo, and P. A. Beerel, "TriLock: IC Protection with Tunable Corruptibility and Resilience to SAT and Removal Attacks," *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1329–1334, 2022.

[25] Z. Han, A. Dixit, S. Patnaik, and J. Rajendran, "STATION: State Encoding-Based Attack-Resilient Sequential Obfuscation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 2888–2901, 2024.

[26] P. Chakraborty, J. Cruz, A. Alaql, and S. Bhunia, "SAIL: Analyzing structural artifacts of logic locking using machine learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3828–3842, 2021, conference Name: IEEE Transactions on Information Forensics and Security. [Online]. Available: https://ieeexplore.ieee.org/document/9478898/?arnumber=9478898

[27] D. Sisejkovic, F. Merchant, L. M. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, "Challenging the Security of Logic Locking Schemes in the Era of Deep Learning: A Neuroevolutionary Approach," *J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 3, pp. 30:1–30:26, 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3431389

[28] A. Alaql, M. M. Rahman, and S. Bhunia, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 8, pp. 1529–1542, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9466931/?arnumber=9466931

[29] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1602–1606, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9539868/?arnumber=9539868

[30] L. L. Mankali, O. Sinanoglu, and S. Patnaik, "INSIGHT: Attacking Industry-Adopted Learning Resilient Logic Locking Techniques Using Explainable Graph Neural Network," *USENIX Security Symposium*, pp. 91–108, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/mankali

Resilient Federated Learning on Embedded Devices with Constrained Network Connectivity

Zihan Li†, Han Liu†, Ao Li†, Ching-hsiang Chan†,
Yevgeniy Vorobeychik†, William Yeoh†, Wenjing Lou‡, Ning Zhang†
† Washington University in St. Louis, MO, USA
‡ Virginia Polytechnic Institute and State University, VA, USA
{tomson.li, h.liu1, ao, c.ching-hsiang, yvorobeychik, wyeoh, zhang.ning}@wustl.edu, wjlou@vt.edu

***Abstract*—Federated learning enables decentralized model training while preserving data privacy. However, since the learning process overlays the physical network infrastructure, the efficiency of learning can be impacted by network connectivity. In this work, we conducted extensive experiments to empirically characterize the impacts and leverage the insights to propose an adaptive federation framework, where clients with limited bandwidth are only prompted to transmit adaptively compressed gradient updates when the gradient similarity score is similar between the local and global models. Our evaluation in simulated environments and on real hardware devices shows bandwidth savings of 60% to 78% compared to state-of-the-art methods.**

I. Introduction

Federated learning (FL) has emerged as a transformative paradigm in decentralized machine learning, allowing distributed devices to collaboratively train a global model while preserving the privacy of local data [1]. This approach has gained widespread attention for its applications in privacy-sensitive domains such as healthcare [2], [3], finance [4], smart devices [5], and personalized services [6], [7]. Federated learning operates as a distributed system of nodes with varying computing power and network connectivity. As an overlay on existing network infrastructure, its success heavily depends on underlying network capabilities.

Existing Solutions and Limitations: Recognizing this challenge, existing works have investigated latency or bandwidth constraint optimization techniques. Protocol-level optimizations [8], [9] focus on timing alignment to mitigate staleness issues due to network latency, whereas model-level techniques [10], [11] focus on reducing the size of the gradient for transmission due to bandwidth constraints. However, all of these strategies are static, which are pre-defined according to the target execution environment and network conditions. In contrast, real-world network environments are often highly dynamic, requiring strategies that can adapt to real-time changes in network connectivity.

Examination of FL Resiliency: To understand the challenges and opportunities, we conducted an empirical study of FL under different network conditions and varying model complexities. Our investigation revealed two key insights that hold valid across different datasets, models, data distributions, and both synchronous and asynchronous FL settings: (1) A moderate level of client update dropouts (approximately 20%) has minimal adverse effects on global model accuracy. (2) In asynchronous FL, model staleness poses an even more negative impact on accuracy compared to client dropouts.

Our Solution – AdaFL: This paper proposes AdaFL, a novel adaptive FL framework that operates in real-time to enhance communication efficiency while maintaining model performance. Based on our first insight, we propose an adaptive node selection algorithm to optimize on client selection. Instead of requesting updates from all available clients, AdaFL adaptively evaluates the utility of each update on the global model's learning trajectory and selectively chooses updates from clients with meaningful updates and sufficient network bandwidth. This approach minimizes communication overhead by prioritizing impactful updates while preserving model performance. Based on our second insight, we conclude that a timely model update from server to client is crucial for model accuracy. Hence, to ensure the timeliness of the update, we propose to adjust the gradient compression rate dynamically. This approach minimizes unnecessary communication overhead for clients with less impactful updates and more constrained bandwidth while preserving sufficient information from clients that provide meaningful contributions, allowing both timely and informational model updates.

Implementation and Evaluation: We have conducted an extensive evaluation of AdaFL across different datasets and models. The evaluation results demonstrate that AdaFL outperforms state-of-the-art (SOTA) FL methods up to 30% on testing accuracy while reducing 60% to 78% communication cost. Moreover, AdaFL incurs a negligible computational overhead, with an average expansion of 0.05% CPU cycles.

Contributions: This paper makes the following contributions:

- **Empirical Study of FL Resilience**: We empirically showed that FL can remain robust under asynchronous updates and client dropouts. This finding underscores the opportunity for optimizing communication without compromising accuracy.
- **Design of AdaFL**: We propose a novel FL framework that selectively aggregates client updates and adapts the clients' gradient compression rates based on the utility of their contributions and their network connectivity.
- **Evaluation**: We demonstrate that AdaFL outperforms state-of-the-art (SOTA) FL methods.

II. Related Work

Existing works generally address network constraints from either protocol-level or model-level perspectives.

Protocol-level: Existing protocol optimizations primarily target reducing wait time from network latency. Zhu et al. [8] parallelize communication and training, delaying server aggregation such that client-side wait time for *stragglers* is minimized. Zhang et al. [9] reduces server aggregation blocking, enabling global model updates during aggregation. FedAT [12] introduces a tiering module that groups clients by network performance for tier-based aggregation. However, these approaches focus solely on improving latency but often neglect model performance implications. In practice, reduced latency doesn't always translate to better model quality. AdaFL addresses this limitation by jointly optimizing both objectives—minimizing latency while maintaining model performance.

Model-level: At the model level, reducing transmitted data is a common strategy, achieved through gradient quantization [11], [13] or compression [10], [14], [15]. Another approach encodes local models into lower-dimensional representations to minimize transmission overhead [16]. Xiong et al. [17] replace full model updates with synthesized datasets, while Wang et al. [18] use progressive learning to shrink model size. Though these methods mitigate communication bottlenecks to some extent, they often assume the network condition is static, whereas, in practice, network conditions are highly dynamic.

III. Exploration of FL Network Resiliency

A. Modeling Impacting Factors on FL

In FL, client-server interactions involve downloading the global model and uploading local updates. In each communication round, clients download the global model from the server and perform model training. After local training, clients upload their updated local models to the server for aggregation. Varying network conditions can introduce latency on the uplink, downlink, or both.

We formalize the impact of network conditions on FL performance by analyzing its underlying optimization process. The optimization problem in FL is:

$$\min_{w} \quad F(w) = \sum_{i=1}^{N} p_i F_i(w), \tag{1}$$

where $w \in \mathbb{R}^d$ represents the global model parameters, N is the total number of participating clients, $F_i(w)$ is the local loss function for client i, and $p_i \geq 0$ is the weighting coefficient assigned to client i. Typically, in synchronous settings, p_i is equal to w_i, which is defined as $w_i = n_i/n$, where n_i denotes the number of data samples held by client i, and $n = \sum_{i=1}^{N} n_i$ represents the total number of data samples across all clients.

During each communication round n, client i computes and sends its local gradient g_i^n to the server. The server then performs aggregation using the following equation:

$$G^{n+1} = G^n + \sum_{i=1}^{N} p_i g_i^n, \tag{2}$$

where G^n is the global gradient at round n, and G^{n+1} is the updated global gradient after aggregation.

To assess the impact of network delay, we analyze two scenarios: synchronous FL and asynchronous FL. Previous research has predominantly focused on synchronous FL [19], [20], [21], where the server waits for all clients to submit their local updates before aggregating them to update the global model. Let Ψ_i denote the computation time for client i, while Υ_i^u and Υ_i^d represent the uplink and downlink network delays for client i, respectively. Under the synchronous protocol, the total time required for one global iteration is:

$$T_{\text{sync}} = \max_i (\Psi_i + \Upsilon_i^u + \Upsilon_i^d). \tag{3}$$

Thus, the synchronization time is constrained by the slowest client due to network delays. Furthermore, if an update from a particular client fails to reach the server, *e.g.*, in cases of unreliable connections, the server may indefinitely wait for this update. The server can instead impose a maximum wait time, dropping any delayed updates beyond this threshold; however, this approach may reduce the accuracy of the global model.

For asynchronous FL, we denote T_i^n as the total time including the training time for performing its local work and sending it to the server aggregation, thus we have $T_i^n = \Psi_{i,n} + \Upsilon_{i,n}^u + \Upsilon_{i,n}^d$, we assume a typical asynchronous setting where the server will wait for Δt^n to perform the aggregation [22], [23]. Note that if the server sets the round time $\Delta t^n = \max_i T_i^n$, the aggregation occurs after receiving contributions from all clients; on the client, if $\Delta t^n = \min_i T_i^n$, it will become fully asynchronous, where every client update results in a server model update.

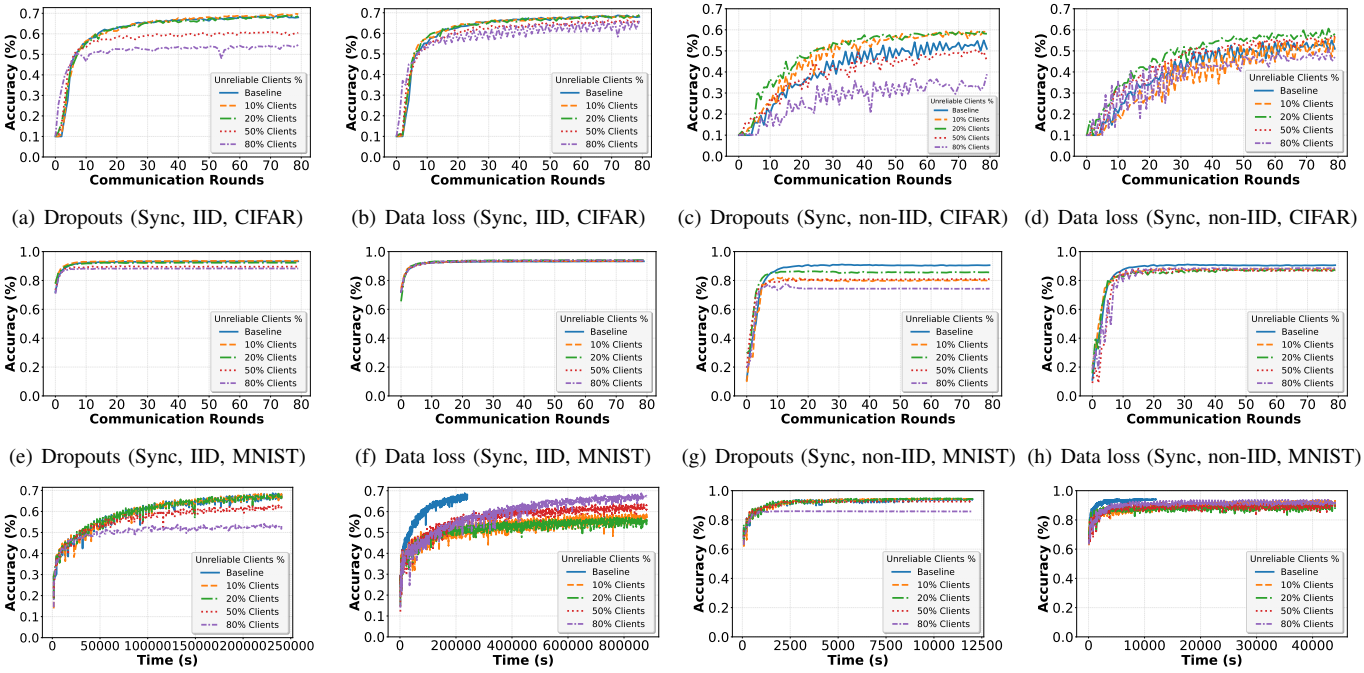
Comparing T_i^n with Δt^n indicates whether a client is participating in the optimization round or not. When $\mathbb{I}(T_i^n \leq \Delta t^n) = 1$, the local work of client i is used to create the new global model θ^{n+1}, while client i does not contribute when $\mathbb{I}(T_i^n \leq \Delta t^n) = 0$. To account for their scenarios of asynchronous updates, we introduce the aggregation weight $d_i(n)$ corresponding to the weight given by the server to client i at optimization round n, we can then denote it as:

$$p_i(n) = \mathbb{I}(T_i^n \leq \Delta t^n) w_i, \tag{4}$$

where w_i follows the same definition as in synchronous FL, generally referring to the proportion of data samples owned by client i. Under asynchronous aggregation, the global update is performed using the following scheme:

$$G^{n+1} = G^n + \sum_{i=1}^{N} \mathbb{I}(T_i^n \leq \Delta t^n) p_i g_i^n. \tag{5}$$

From the above analysis, network conditions primarily influence the client dropout rate in synchronous settings and the staleness of client updates in asynchronous settings.



(a) Dropouts (Sync, IID, CIFAR) (b) Data loss (Sync, IID, CIFAR) (c) Dropouts (Sync, non-IID, CIFAR) (d) Data loss (Sync, non-IID, CIFAR)

(e) Dropouts (Sync, IID, MNIST) (f) Data loss (Sync, IID, MNIST) (g) Dropouts (Sync, non-IID, MNIST) (h) Data loss (Sync, non-IID, MNIST)

(i) Dropouts (Async, IID, CIFAR) (j) Staleness (Async, IID, CIFAR) (k) Dropouts (Async, IID, MNIST) (l) Staleness (Async, IID, MNIST)

Fig. 1. Testing accuracy of ResNet-50 on CIFAR-10 and CNN on MNIST under different network conditions, data distributions, and FL protocols.

B. Empirical Measurement of Network Impact on FL

To understand how these factors influence client-server interactions and the overall training process, we simulated diverse link latencies and bandwidth constraints. Our examination covered both synchronous and asynchronous FL protocols to analyze their behavior under challenging network conditions. Additionally, we compared performance across different data distribution settings, including IID (Independent and identically distributed) and non-IID scenarios, to highlight how data heterogeneity interacts with network dynamics.

Synchronous and Asynchronous FL: The impact of network conditions on FL performance differs between synchronous and asynchronous protocols. In synchronous FL, high latency causes intermittent client updates to reach the server (*data loss*), disrupting aggregation and delaying global model updates. In asynchronous FL, high latency exacerbates staleness, where client updates are based on outdated global model versions (*staleness*). In our experimental setup, high-latency clients in synchronous FL update the server every other communication round, while in asynchronous settings, they update at a rate $3\times$ slower than other clients, contributing only after others have updated three times.

Network Conditions: We primarily emulate various networks from the perspectives of bandwidth and link conditions.

Bandwidth: The effect of link bandwidth is consistent across both protocols: insufficient bandwidth prevents clients from transmitting updates (*dropouts*), reducing data diversity and contributing to the global model.

Link Conditions: We consider uplink and downlink collectively, as their impact on model updates is equivalent. Specifically, we examine two network conditions: link latency and link bandwidth. High latency delays client updates, slowing the server's aggregation process. Insufficient bandwidth prevents clients from transmitting updates, excluding their contributions from training. We use simulated network data from the ns3 network simulator [24] to model suboptimal network links for clients in both synchronous and asynchronous FL scenarios.

Data Distributions: We consider both IID and non-IID data distributions, following the non-IID setting described in [19]. IID data simplifies training by ensuring consistent client contributions and serves as a baseline for ideal conditions. However, real-world applications, such as healthcare and mobile services, typically involve non-IID data, where heterogeneous client data can slow convergence, reduce accuracy, and amplify communication bottlenecks and dropouts.

Testing Models and Platforms: We implemented a fully synchronous FL protocol using FedAvg [19] and a fully asynchronous FL protocol using FedAsync [22], both commonly used as baselines in existing works. We selected 10 clients and varied the proportion of unreliable clients in each experiment. For model training, we used ResNet-50 [25] on the CIFAR-10 [26] dataset and another typical CNN model used as a baseline in prior studies [27] on the MNIST [28] dataset. The ResNet-50 model is a deep architecture with 50 layers and residual connections. The CNN model consists of two 5×5 convolution layers: the first layer has 20 output channels, and the second has 50, each followed by 2×2 max pooling. Evaluations were conducted on a system with an 18-core Intel® Core™ i9-7980XE CPU, 64 GB DRAM, and an NVIDIA RTX 3090 GPU.

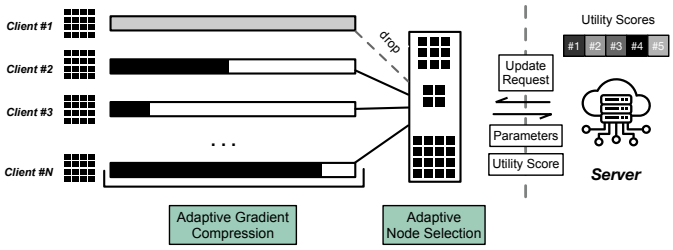


Fig. 2. Overview of AdaFL.

Results: Figure 1 shows the measurement results from our empirical study, with all experiments repeated 10 times to reduce randomness. Figure 1(a) to Figure 1(h) shows that 10% to 20% dropout and data loss had a negligible impact on overall performance in most cases. However, data loss exhibited a more pronounced negative effect due to the noise introduced by stragglers. In terms of model and dataset complexity, the impact of a higher proportion of stragglers was more significant, as shown in Figure 1(a),(e). Interestingly, in the CIFAR-10 experiments under non-IID settings, scenarios with 10% to 20% stragglers outperformed the baseline, as shown in Figure 1(c). In asynchronous settings, staleness consistently led to lower accuracy and convergence rates across all conditions than dropouts. Due to staleness, it requires a much longer time to reach a similar accuracy compared to dropouts, as shown in Figure 1(i) to Figure 1(l).

Insights: We derived two key insights:

- A small amount of client dropout and data loss have a minimal negative impact on the overall testing accuracy and convergence rate in synchronous and asynchronous FL settings. This leaves headroom for the optimization of client selection to reduce communication costs. Notably, for non-IID data distributions, the client participation rate has a limited effect on overall testing accuracy. This observation is consistent with the findings of Li et al. [29], which proves that, under non-IID settings, the convergence rate weakly replies to the client participation ratio. In a real-world deployment, client data is naturally non-IID, indicating FL robustness to partial client participation under such data distributions.
- In asynchronous FL, the effect of staleness (outdated updates) has a more significant impact on overall testing accuracy compared to client dropouts, highlighting the sensitivity of asynchronous protocols to temporal misalignment. Chen et al. [30] also demonstrate that staleness can increase the variances and affect the convergence of the global model. This motivates us to adjust gradient granularity due to network constraints adaptively.

IV. Design

Overview: Our empirical study reveals that FL exhibits a degree of tolerance to lower client participation rates, with no significant negative impact on overall training performance. However, staleness has a more detrimental effect on model accuracy than client dropouts. Building on these insights, we propose a utility and network connectivity guided FL framework, AdaFL, as shown in Figure 2. Unlike traditional optimization approaches that rely on link conditions, AdaFL leverages gradient similarity to calculate a utility score and adaptively select clients for contributing to the global model. We eliminate unnecessary communication costs associated with lower-impact contributions by prioritizing clients with higher utility scores. To further minimize communication overhead and mitigate staleness effect, AdaFL incorporates deep gradient compression [10], dynamically adjusting the compression rate based on feedback from utility score ranking. This combined approach optimizes communication efficiency while maintaining robust model performance.

Algorithm 1: Adaptive Node Selection

Input: Global model gradient vector $g_G \in \mathbb{R}^d$, Set of local gradients $\{g_i\}_{i=1}^N$, where $g_i \in \mathbb{R}^d$ for client i, Number of clients to select $K \in \mathbb{N}$, where $1 \leq K \leq N$, Utility threshold $\tau \in [0, 1]$
Output: Selected client set $\mathcal{C}_{\text{selected}} \subseteq \{1, \ldots, N\}$
Initialize: $\mathcal{C}_{\text{selected}} \leftarrow \emptyset$

Utility Score Computation:
for $i \leftarrow 1$ **to** N **do**
  Compute utility score: $S_i \leftarrow f(B_i^{down}, B_i^{up}, U(g_i, \hat{g}))$
end

Client Filtering:
$\mathcal{C}_{\text{filtered}} \leftarrow \{i \in \{1, \ldots, N\} : S_i \geq \tau\}$

Client Ranking and Selection:
Sort $\mathcal{C}_{\text{filtered}}$ by S_i in descending order
$K' \leftarrow \min(K, |\mathcal{C}_{\text{filtered}}|)$
$\mathcal{C}_{\text{selected}} \leftarrow$ First K' elements of sorted $\mathcal{C}_{\text{filtered}}$

Return: $\mathcal{C}_{\text{selected}}$
Subject to:
- $|\mathcal{C}_{\text{selected}}| \leq K$
- $\forall i \in \mathcal{C}_{\text{selected}} : S_i \geq \tau$
- $\forall i \in \mathcal{C}_{\text{selected}}, j \notin \mathcal{C}_{\text{selected}} : S_i \geq S_j$

Adaptive Node Selection: Our extensive experiments reveal that occasionally excluding certain clients does not negatively impact convergence accuracy, especially under non-IID settings. This observation suggests that a smaller participation ratio can be set to alleviate the straggler's effect without hurting model accuracy. Building on this insight, we propose an adaptive node selection algorithm to optimize client participation. In each training round, our approach dynamically prioritizes selecting the most impactful clients based on their contributions to the global model. Upon receiving the global model from the server, the client interrupts its current training process to calculate a utility score. This score determines whether the client should update its local model to align with the global model or continue with its existing training state, minimizing additional communication and computation overhead. The utility score S_i for each client i is defined as:

$$S_i = f(B_i^{down}, B_i^{up}, U(g_i, \hat{g})) \tag{6}$$

Here, S_i is a function of bandwidth (B_i^{down},B_i^{up}) and similarity metric U, which measures the similarity between the gradient of the local model g_i and the gradient of the global model $\hat{g}$ from previous round. We use cosine similarity, commonly used in existing ML works [31], [32], to measure the similarity between two models. Other metrics, such as L^2 norm and Euclidean distance [33], can also be used as a similarity metric to measure the local gradient importance toward the global

model. A high utility score between local and global gradients indicates alignment, suggesting that a client's update will likely contribute positively to the global model's convergence. Conversely, a low utility score represents misalignment, indicating potential noise or convergence impediments. Based on this utility score, we establish a ranking system that prioritizes clients whose gradient directions align more closely with the global gradient, as their updates are more likely to enhance model performance. Furthermore, because cosine similarity is directionally sensitive, it captures both positive and negative gradient directions. This directional awareness allows it to remain robust even during oscillations caused by high learning rates (typically in SGD), ensuring consistent detection of essential gradient patterns. To effectively manage heterogeneous data distributions across the client network and mitigate the impact of these oscillations, AdaFL integrates a warm-up stage designed to stabilize directional oscillations early in training. During these crucial early rounds, we maintain equal participation from all clients, enabling the global model to adapt gradually to diverse data patterns without premature specialization. Following this warm-up period, AdaFL transitions to a selective participation model, where only top-ranked clients—determined by the utility score—contribute to each training round. This adaptive selection strategy strikes an optimal balance between computational efficiency and model convergence quality. This adaptive selection algorithm is detailed in Algorithm 1.

Adaptive Gradient Compression: The second component addresses the critical challenge of communication overhead in FL through an adaptive compression strategy. While gradient compression effectively reduces communication costs, it must be carefully calibrated to prevent convergence degradation. We introduce a dynamic compression mechanism that changes the compression ratio based on the utility score (equation 6), preserving essential information while minimizing bandwidth consumption. Building upon deep gradient compression (DGC) [10], our approach selectively transmits significant updates while accumulating minor gradients locally. The compression ratio is adjusted based on the utility score: clients with higher utility scores receive less compression to preserve important information, while updates from less critical clients are compressed more aggressively. This selective transmission strategy reduces bandwidth requirements while maintaining model convergence through local gradient accumulation. To enhance compression effectiveness, we integrate two key DGC components: momentum correction and local gradient clipping. Momentum correction harmonizes sparse updates with dense updates temporally, mitigating convergence issues from delayed transmissions. Local gradient clipping, applied pre-accumulation, prevents gradient explosion and maintains training stability under high compression rates. During initial warm-up rounds, we maintain low compression rates across all clients to ensure robust model initialization. As the training progresses, compression rates adapt continuously based on the utility score, optimizing the balance between communication efficiency and update significance.

TABLE I
SYNCHRONOUS FL EVALUATION RESULTS

Synchronous FL Methods	# Clients	Particip. Rate	Update Freq.	Cost Reduc. (%)	Gradient Size	Compress. Ratio	Top 1 Accuracy (IID / non-IID)
FedAvg	10	0.5	400	-50%	1.64MB	1x	MNIST: 93.6% / 86.88% CIFAR-100: 62.34% / 55.83%
FedAdam	10	0.5	400	-50%	1.64MB	1x	MNIST: 93.06% / 86.62% CIFAR-100: 61.76% / 54.92%
FedProx	10	0.5	400	-50%	1.64MB	1x	MNIST: 93.31% / 85.72% CIFAR-100: 62.02% / 54.89%
SCAFFOLD	10	0.5	400	-50%	1.64MB	1x	MNIST: 93.13% / 90.35% CIFAR-100: 61.80% / 57.73%
AdaFL	10	Adaptive	233	-70.88%	8 - 420KB	210x - 4x	MNIST: 93.43% / 87.47% CIFAR-100: 61.86% / 56.29%

TABLE II
ASYNCHRONOUS FL EVALUATION RESULTS

Asynchronous FL Methods	# Clients	Particip. Rate	Update Freq.	Cost Reduc.	Gradient Size	Compress. Ratio	Top 1 Accuracy (IID / non-IID)
FedAsync	10	0.5	400	-50%	1.64 MB	1x	MNIST: 93.25% / 85.69% CIFAR-100: 61.39% / 54.28%
FedBuff	10	0.5	400	-50%	1.64 MB	1x	MNIST: 93.72% / 88.02% CIFAR-100: 62.21% / 56.72%
AdaFL	10	Adaptive	172	-78.5%	16KB - 400KB	105x - 4x	MNIST: 93.58% / 89.11% CIFAR-100: 62.14% / 57.69%

V. EVALUATION

We evaluated AdaFL with other FL methods to answer the following three questions:

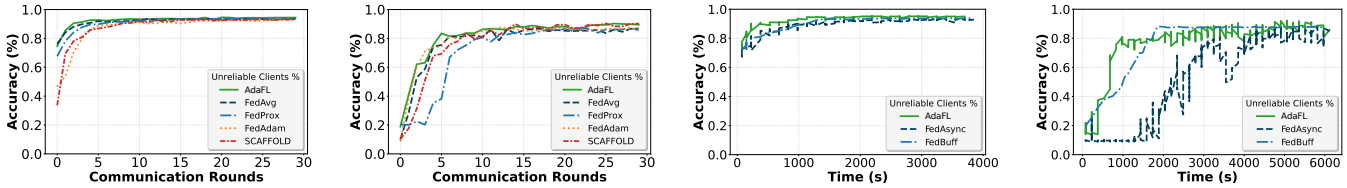
Q1. How does AdaFL perform (testing accuracy rate) compared with other optimization methods?

Q2. Can AdaFL significantly reduce communication cost while ensuring training performance?

Q3. What is the overhead of AdaFL?

Experimental Settings: We compared AdaFL to existing state-of-the-art approaches in both synchronous and asynchronous FL. In the synchronous FL domain, we benchmarked our method against FedAvg [19], FedAdam [34], FedProx [20], and SCAFFOLD [21]. Under synchronous context, AdaFL used $top - k$ topology setting, where we perform weights averaging from k clients for each communication round. For asynchronous FL comparisons, we evaluated against FedAsync [22] and FedBuff [35]. Under asynchronous context, AdaFL adapts fully asynchronous FL, where the server upgrades its global model each time it receives a gradient update. To ensure fair comparisons, we maintained consistency across all experiments using the same CNN architecture and MNIST dataset from our empirical study. We chose $r_p = 0.5$ as our participation rate out of 10 clients and strictly adhered to other hyperparameters for each baseline method while conducting all evaluations on our established experimental platform. We conduct all experiments under a fixed bandwidth to observe the reduction cost. To further validate the effectiveness of AdaFL on more complex models and datasets, we evaluated it using VGG-Net [36] on CIFAR-100 [26]. Additionally, we conducted experiments with 20 to 100 clients to assess its scalability, demonstrating its robustness across varying client participation levels.

Effectiveness: Figure 3 plots the model performance of evaluated frameworks. It can be observed that AdaFL has a higher convergence rate than existing works, as our training process is dynamically adapted and guided by the utility score.



(a) Sync, IID, MNIST (b) Sync, non-IID, MNIST (c) Async, IID, MNIST (d) Async, non-IID, MNIST

Fig. 3. Testing accuracy of CNN on MNIST for Synchronous and Asynchronous FL Protocols.

All frameworks we compared have a fixed participation rate, while ours are adjusted dynamically where $k \leq 5$. As shown in Figure 3, our learning curve is higher and more stable, especially under non-IID settings. In Figure 3(b), AdaFL has higher overall testing accuracy and a more stable learning curve from round 1 to round 5, showing the effectiveness of AdaFL under non-IID data distributions. In addition, AdaFL has a significantly higher convergence rate for asynchronous FL under non-iid settings than other frameworks, as shown in Figure 3(d). At $T = 1000s$, AdaFL reached around 80% testing accuracy where FedAsync is around 10% and FedBuff is around 50%. FedBuff has a lower noise in non-IID training due to its optimization of the buffer for weight averaging. However, AdaFL still has a higher convergence rate at the beginning of the training and can converge to a higher testing accuracy of 89.11% compared to 88.02% for FedBuff.

Answer to Q1. Under both IID and non-IID data distributions, AdaFL can outperform other methods with a higher convergence rate and testing accuracy. Specifically, under non-IID data distributions, the accuracy improvements ranged from 5% up to 70%. The results indicate the importance of the utility score guided training, especially under non-IID settings.

Communication Reduction: Table I and Table II summarizes the communication cost reduced from AdaFL. For all methods we compared, we set a fixed participation rate of $r_p = 0.5$ for each communication round, while ours has a dynamic participation rate of $r_p \leq 0.5$ where r_p is a positive value. As a result, all compared methods have a fixed update frequency, where clients update 400 times to the server for global model updates. We consider the ideal update frequency to be 800 times when all clients participate. In synchronous settings, AdaFL achieved an average of 233 update frequencies to the server, which reduced 42% unnecessary updates compared to other methods and reduced 70.88% overall communication costs. Similarly, in asynchronous settings, AdaFL had an average of 172 update frequencies, which reduced 57% of updates compared to others and reduced 78.5% overall communication costs. On the other hand, we further reduced communication costs by applying gradient compression, which AdaFL reached a compression ratio of up to 210x. This allows the server and clients to transmit gradient updates faster, which is beneficial under asynchronous contexts where the server can have a higher convergence rate from more frequent updates.

Answer to Q2. AdaFL reduces communication costs significantly in two aspects: update frequencies and gradient size. While the communication cost is reduced, AdaFL performance is still robust, as discussed previously.

Overhead: We break down the overhead into two individual components: overhead on utility score calculation and gradient compression. We performed an ablation study using a Raspberry Pi cluster to validate our framework's adaptability across heterogeneous devices. Our experimental setup comprised a ten-node cluster configuration, where we trained a CNN model on the MNIST dataset. We utilized *perf*, a Linux performance profiling tool, and recorded CPU cycle counts with and without AdaFL to measure the performance overhead induced by the two components. With a baseline cycle count of 8,589,175,469,216, our experimental results show a maximum of cycle counts for utility score calculation is 8,783,108,492,282, which is only around 0.05% more than the baseline. On the other hand, the performance overhead added for gradient compression is larger than the utility score calculation. However, clients can save computational resources from training and gradient compression based on the utility score. If the utility score for a client is low, the client can halt training until it can provide meaningful gradients to the server (i.e., wait for the next global update). As a result, the cost reduction from Table I and II also shows computational cost reduction. The overall overhead is negligible compared to the cost reduction benefit from the utility score calculation.

Answer to Q3. AdaFL adds additional overhead more significantly from gradient compression than utility score calculation. However, the computational cost reduction benefit from the adaptive client selection process significantly balances the overhead from gradient compression computation. Hence, the overall overhead is negligible.

VI. Conclusion

This paper empirically studies factors affecting FL performance, revealing resilience to client dropouts. Leveraging this, we introduce AdaFL for online adaptation. Evaluations show AdaFL outperforms SOTA FL methods by up to 30% in testing accuracy while ensuring scalability and robustness in dynamic environments. Additionally, AdaFL incurs negligible computational overhead.

Acknowledgment

We thank the reviewers for their valuable feedback. This work was partially supported by the NSF (CNS-2154930, CNS-2229427), ARO (W911NF-24-1-0155), and ONR (N00014-24-1-2730, N00014-24-12663).

REFERENCES

[1] N. Wang, Y. Xiao, Y. Chen, N. Zhang, W. Lou, and Y. T. Hou, "Squeezing more utility via adaptive clipping on differentially private gradients in federated meta-learning," in *Proceedings of the 38th Annual Computer Security Applications Conference*, pp. 647–657, 2022.

[2] R. S. Antunes, C. André da Costa, A. Küderle, I. A. Yari, and B. Eskofier, "Federated learning for healthcare: Systematic review and architecture proposal," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 4, pp. 1–23, 2022.

[3] Y. Chang, H. Liu, E. Jaff, C. Lu, and N. Zhang, "Sok: Security and privacy risks of medical ai," *arXiv preprint arXiv:2409.07415*, 2024.

[4] J. Morgan, "Federated learning meets blockchain." https://www.jpmorgan.com/technology/news/federated-learning-meets-blockchain, 2024. Accessed: 2024-11-18.

[5] L. Babun, Z. B. Celik, P. McDaniel, and A. S. Uluagac, "Real-time analysis of privacy-(un) aware iot applications," *Proceedings on Privacy Enhancing Technologies*, 2021.

[6] H. Chen, J. Ding, E. W. Tramel, S. Wu, A. K. Sahu, S. Avestimehr, and T. Zhang, "Self-aware personalized federated learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 20675–20688, 2022.

[7] H. Liu, Y. Wu, Z. Yu, and N. Zhang, "Please tell me more: Privacy impact of explainability through the lens of membership inference attack," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 4791–4809, IEEE, 2024.

[8] L. Zhu, H. Lin, Y. Lu, Y. Lin, and S. Han, "Delayed gradient averaging: Tolerate the communication latency for federated learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29995–30007, 2021.

[9] F. Zhang, X. Liu, S. Lin, G. Wu, X. Zhou, J. Jiang, and X. Ji, "No one idles: Efficient heterogeneous federated learning with parallel edge and server computation," in *International Conference on Machine Learning*, pp. 41399–41413, PMLR, 2023.

[10] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[11] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in neural information processing systems*, vol. 30, 2017.

[12] Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala, "Fedat: A high-performance and communication-efficient federated learning system with asynchronous tiers," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–16, 2021.

[13] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," *Advances in neural information processing systems*, vol. 30, 2017.

[14] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[15] H. Tang, C. Yu, X. Lian, T. Zhang, and J. Liu, "Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression," in *International Conference on Machine Learning*, pp. 6155–6165, PMLR, 2019.

[16] J. Li and H. Huang, "Resolving the tug-of-war: a separation of communication and learning in federated learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[17] Y. Xiong, R. Wang, M. Cheng, F. Yu, and C.-J. Hsieh, "Feddm: Iterative distribution matching for communication-efficient federated learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16323–16332, 2023.

[18] H.-P. Wang, S. Stich, Y. He, and M. Fritz, "Progfed: Effective, communication, and computation efficient federated learning by progressive training," in *International Conference on Machine Learning*, pp. 23034–23054, PMLR, 2022.

[19] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[20] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[21] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International conference on machine learning*, pp. 5132–5143, PMLR, 2020.

[22] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[23] Y. Fraboni, R. Vidal, L. Kameni, and M. Lorenzi, "A general theory for federated optimization with asynchronous and heterogeneous clients updates," *Journal of Machine Learning Research*, vol. 24, no. 110, pp. 1–43, 2023.

[24] E. Ekaireb, X. Yu, K. Ergun, Q. Zhao, K. Lee, M. Huzaifa, and T. Rosing, "ns3-fl: Simulating federated learning with ns-3," in *Proceedings of the 2022 Workshop on ns-3*, pp. 97–104, 2022.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[26] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[27] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE INFOCOM 2020-IEEE conference on computer communications*, pp. 1698–1707, IEEE, 2020.

[28] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[29] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv preprint arXiv:1907.02189*, 2019.

[30] M. Chen, B. Mao, and T. Ma, "Efficient and robust asynchronous federated learning with stragglers," in *International Conference on Learning Representations*, 2019.

[31] H. Qin, S. Rajbhandari, O. Ruwase, F. Yan, L. Yang, and Y. He, "Simigrad: Fine-grained adaptive batching for large scale training using gradient similarity measurement," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20531–20544, 2021.

[32] M. Fang, J. Liu, N. Z. Gong, and E. S. Bentley, "Aflguard: Byzantine-robust asynchronous federated learning," in *Proceedings of the 38th Annual Computer Security Applications Conference*, pp. 632–646, 2022.

[33] Z. Wang, Q. Hu, X. Zou, P. Hu, and X. Cheng, "Can we trust the similarity measurement in federated learning?," *IEEE Transactions on Information Forensics and Security*, 2025.

[34] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[35] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *International Conference on Artificial Intelligence and Statistics*, pp. 3581–3607, PMLR, 2022.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

GSAcc: Accelerate 3D Gaussian Splatting via Depth Speculation and Gaussian-centric Rasterization

Mengtian Yang, Yipeng Wang, Chieh-Pu Lo, Xiuhao Zhang, Sirish Oruganti and Jaydeep P. Kulkarni
The University of Texas at Austin
{mengtian.yang, yipeng.wang, kcplo, zhxhao007, sirishoruganti}@utexas.edu, jaydeep@austin.utexas.edu

***Abstract*—3D Gaussian Splatting (3DGS) has emerged as a promising real-time photorealistic radiance field rendering technique. Existing GPU and hardware accelerators face limitations due to insufficient parallelism in sequential rendering pipeline stages and the memory overhead associated with interim results. This paper presents GSAcc, a hardware accelerator co-designed with dataflow to render compressed 3DGS models on edge platforms efficiently. GSAcc enhances 3DGS rendering performance through several key innovations. First, it introduces Gaussian depth speculation, parallelizing preprocessing and sorting tasks. Second, GSAcc adopts a Gaussian-centric dataflow that interleaves preprocessing and rasterization, allowing all rendering steps to execute concurrently without storing intermediate results. Finally, it employs dedicated hardware acceleration to address sorting and rasterization bottlenecks within the optimized dataflow. We implemented and synthesized GSAcc using Intel16 PDK and evaluated its performance on real-world 3DGS scenes. Compared with desktop GPUs, GSAcc achieves up to 1.66×10^4x Power-Performance-Area (PPA) improvement as well as 48.7x energy savings. Additionally, GSAcc outperforms the state-of-the-art hardware accelerator GSCore with up to 2.3x PPA improvement and 2.9x energy savings.**

***Index Terms*—3D Gaussian Splatting, Software/Hardware Co-design, Accelerator Architecture, Radiance Field Rendering**

I. Introduction

Neural Radiance Fields (NeRFs) [1] enable photorealistic image synthesis from novel viewpoints by learning continuous volumetric scene representations from images. Still, they suffer from significant computational demands and very high rendering latency. 3D Gaussian Splatting (3DGS) [2] has recently emerged as a substantial advancement in radiance field rendering, explicitly addressing the challenge of rendering speed and achieving higher image quality. This technique utilizes the mathematical characteristics of Gaussian functions to construct continuous and differentiable representations of 3D scenes. By projecting these 3D Gaussians onto image planes, the method can generate high-quality images with smoother transitions and fewer artifacts than traditional NeRF approaches. Additionally, when combined with advanced compression techniques [3], 3DGS achieves impressive rendering speeds on GPU hardware while maintaining a relatively compact model size. This balance of efficiency and quality makes it a promising solution for real-time radiance field rendering applications.

Despite its promising capabilities, 3DGS has substantial challenges in computational efficiency and memory requirements when deployed on AR/VR platforms, which demand real-time rendering performance within limited power and memory constraints. The original 3DGS algorithm contains three interdependent steps, making it difficult to parallelize and pipeline the processes effectively. Additionally, each step requires the computation of intermediate results for millions of Gaussians, which must be stored for subsequent processing. This necessity leads to significant memory overhead, further complicating the deployment of 3DGS in resource-constrained AR/VR environments. This paper introduces GSAcc, a compressed 3DGS accelerator designed to optimize the 3DGS under the edge platform.

In summary, this paper makes the following contributions:

- First, we analyze the rendering pipeline of 3DGS in a real-time model viewer and propose a depth speculation technique based on previous frames to parallelize sorting.
- Second, we introduce a Gaussian-centric dataflow that interleaves pre-processing and rasterization, eliminating the need for storing intermediate results.
- Third, we design and implement GSAcc, a hardware accelerator that supports the optimized dataflow. GSAcc achieves up to $1.66 \times 10^4\times$ PPA improvement and 48.7× energy savings compared with GPU platform and up to 2.3× PPA improvement and 2.9× energy savings compared with state-of-the-art accelerator GSCore [4].

II. Background

In this section, we will introduce the background of 3DGS. We will first explain how to store scene information in 3D Gaussians and then illustrate how to render a scene from the model.

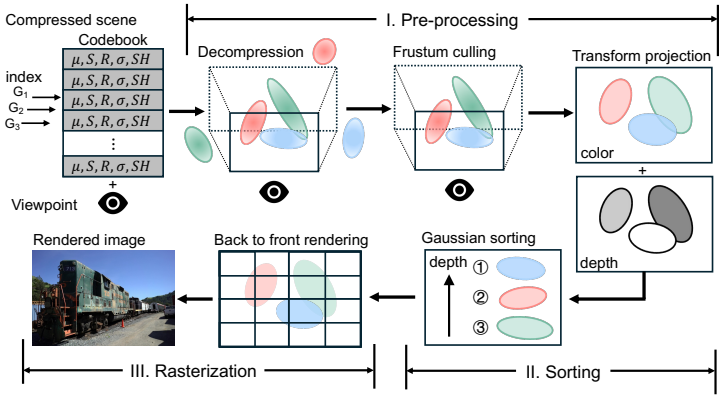


Fig. 1: The Pre-processing, Sorting, and Rasterization pipeline for compressed 3D Gaussian Splatting model rendering.

A. 3DGS Model with Compression

In a 3DGS model, scenes are explicitly represented as ensembles of three-dimensional Gaussian primitives. Each Gaussian has five key attributes: position $\boldsymbol{\mu}$ represents the center of the Gaussian, scale $\boldsymbol{S}$ describes the size of the Gaussian along each axis,

rotation $\boldsymbol{R}$ specifies the orientation of the Gaussian, opacity $\boldsymbol{\sigma}$ indicates the transparency level, and color $\boldsymbol{SH}$ defines the color characteristics encoded using sphere harmonics. Each Gaussian primitive effectively models a localized region of the scene, and the entire scene is constructed to balance geometric fidelity and visual realism by aggregating millions of Gaussians.

Since each scene usually contains many Gaussians, the vanilla 3DGS model is often much larger than a NeRF model for the same scene. To effectively reduce the memory footprint and enhance the computational efficiency of 3DGS models, vector quantization using k-means clustering presents a robust compression strategy [3]. This approach involves partitioning the high-dimensional parameter space of Gaussian primitives, including position, scale, rotation, color, and opacity, into a finite set of representative centroids through the k-means algorithm. By clustering similar Gaussian primitives together, each Gaussian in the original model can be approximated by the nearest centroid from the learned codebook, thereby significantly reducing the number of parameters that need to be stored.

B. Rendering Pipeline

Fig. 1 illustrates the rendering pipeline from the viewpoint of an image within a compressed 3DGS model. This pipeline contains three sequential and interdependent stages: pre-processing, sorting, and rasterization. Intermediate results from each stage are stored in memory for the next stage.

Pre-processing: In the pre-processing stage, each Gaussian is firstly decompressed from the codebook to retrieve its parameters. Once the parameters are obtained, each Gaussian is evaluated to determine whether it falls within the viewing frustum—the region of space potentially visible on the final screen. Based on their depth information, Gaussians outside this frustum are excluded through a process known as frustum culling. Subsequently, the depth of each Gaussian is calculated, and a transformation projection is applied to map the 3D Gaussians onto the 2D screen space. Also, the color of each Gaussian is computed based on its spherical harmonics representation, ensuring accurate color representation during rendering.

Sorting: Following pre-processing, the Gaussians projected onto the 2D screen space are organized based on their depth and color attributes. Since Gaussians closer to the viewpoint can shield those farther away, depth-based sorting is essential to maintain the correct rendering image. All Gaussians are sorted in ascending order of their depth values, from nearest to farthest, and their ordered indices are generated to facilitate the subsequent rendering step.

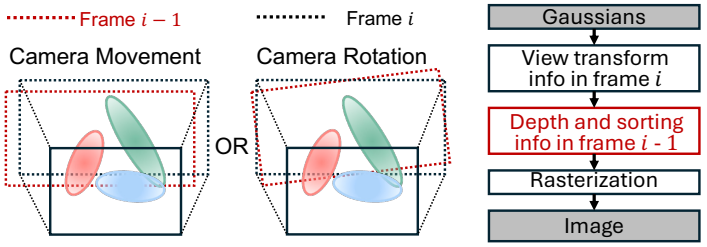
Rasterization: In the rasterization stage, the projected and sorted Gaussians are rendered into the final image. The screen is divided into tiles with 16x16 pixels in each tile, and rendering is performed parallel across tiles to optimize performance. For each pixel, the relevant Gaussians are sampled to determine the pixel opacity (α), and alpha-blending [5] is used to calculate the final color of the pixel. This rendering process proceeds from the back to the front based on the depth information of the Gaussians, ensuring proper occlusion and shielding effects.

III. Motivation and Analysis

This section explores the potential for reusing information from the previous frame to enable parallelism between the sorting step and other processing stages. We then analyze the current rasterization pipeline to identify opportunities for parallelizing pre-processing and rasterization by leveraging data from the last frame. Finally, we examine the tiling process during rasterization to uncover possibilities for optimizing memory access.

A. Last Frame Information Reuse

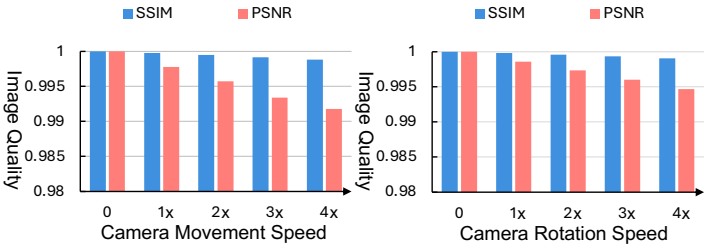
Immersive applications are the primary use case for 3DGS. As illustrated in Fig. 2(a), camera movements and rotations result in changes to the rendered image when viewing a 3DGS model in immersive applications. At a reasonable frames per second (FPS), consecutive frames typically show only minor differences. This temporal coherence suggests that certain information from the previous frame, such as Gaussian depth and sorting results, can be reused in the rendering process for the current frame.



(a) Frame change and render pipeline with last frame depth information



(b) Rendered image with last frame depth and sorting information



(c) Image quality on movement speed (d) Image quality on rotation speed

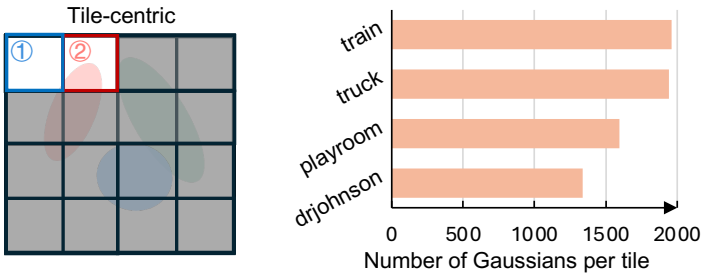
Fig. 2: Image quality analysis leveraging depth information from the previous frame. View transformation is based on current camera data, while Gaussian depth and sorting utilize the previous frame's camera information.

We evaluate the rendered image quality by leveraging depth information and sorting orders from the previous frame under various camera movement and rotation speeds. Fig. 2 (b) shows the rendered image when using the depth from the previous frame. Fig. 2 (c) and (d) present the results of this analysis. The standard

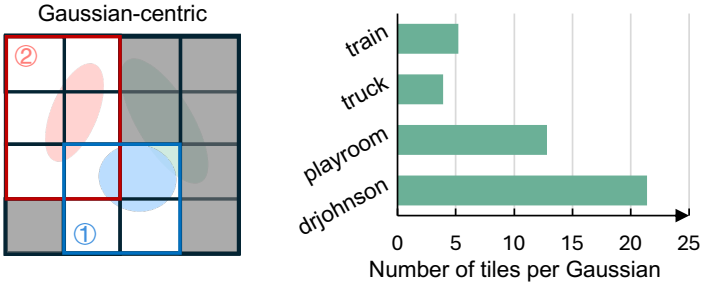
movement speed (0.03 per frame) and rotation speed (0.5 degrees per frame) are based on the interactive viewer from the 3DGS paper [2] and represented as "1x", while a speed of 0 indicates a stationary camera, serving as the ground truth for viewing. To test the robustness of our proposed approach, we accelerate camera movement and rotation speeds up to four times the standard speed. Image quality is evaluated using SSIM and PSNR metrics compared to ground truth. The analysis is performed in four different scenes and averaged. The results show that even at four times the standard camera speed, the degradation of image quality remains within 1% when using Gaussian depth and the sorting results of the previous frame.

B. Tile-centric vs Gaussian-centric Rasterization

The rasterization stage calculates the color of each screen pixel based on the intersecting Gaussians from back to front. Consequently, the rendering can either iterate through all Gaussians for each tile before moving to the next tile, as shown in Fig. 3(a). We refer to it as the *tile-centric* rasterization dataflow. Alternatively, rendering can prioritize Gaussians first and compute colors for all pixels across intersecting tiles, as shown in Fig. 3(b), and we call it *Gaussian-centric* rasterization dataflow.



(a) Tile-centric rasterization and average Gaussians accessed per tile



(b) Gaussian-centric rasterization and average tiles accessed per Gaussian

Fig. 3: Comparison of tile-centric rasterization with Gaussian-centric rasterization.

The right side of Fig. 3 illustrates the average number of Gaussians accessed per tile in the tile-centric dataflow compared to the average number of tiles accessed per Gaussian in Gaussian-centric dataflow. It is evident that each tile requires significantly more Gaussians than the number of tiles each Gaussian interacts with. In tile-centric dataflow, all accessed Gaussians must be ready before rasterizing a tile, aligning with the vanilla 3DGS rendering pipeline in Fig. 1, where pre-processing handles and stores all Gaussian information beforehand. However, by utilizing Gaussian depth and order information from the previous frame, pre-processing and rasterization can be interleaved. In this scenario, Gaussian-centric dataflow is preferable, as it eliminates the need to store interim pre-processing results and simplifies tile management, with each Gaussian typically affecting only a few tiles on average.

C. Tile Stationary Across Gaussians

In Gaussian-centric dataflow, Gaussians are rasterized in ascending order of their depth. As a result, subsequent Gaussians may be positioned close to the current one, leading to overlapping tiles. To minimize DRAM access, the color information for these overlapping tiles should be retained on-chip as much as possible.

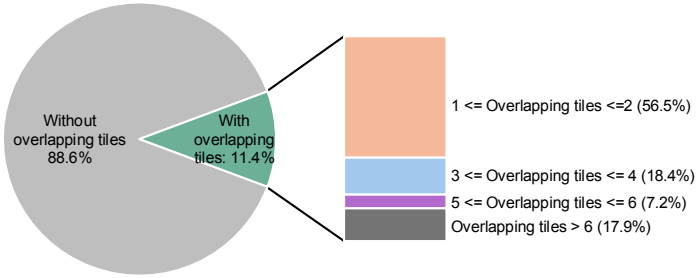


Fig. 4: Overlapping ratio of tiles for current Gaussian and tiles for previous Gaussian with Gaussian-centric rasterization.

Fig. 4 illustrates the ratio of overlapping to non-overlapping tiles between two adjacent Gaussians during rasterization with Gaussian-centric dataflow, averaged across four different benchmarks. The results show that approximately 11.4% of tiles overlap with those of the preceding Gaussian. Additionally, the majority of these overlaps occur on four or fewer tiles, indicating that a relatively small on-chip buffer is sufficient to optimize tile stationary effectively.

IV. Proposed Design

This section illustrates the proposed GSAcc, a 3D Gaussian Splatting accelerator architecture co-designed with depth speculation and Gaussian-centric dataflow. We will first introduce the architecture design and each hardware unit, followed by the proposed fully parallelized dataflow.

A. Overall Architecture

Fig. 5 illustrates the overall architecture of our proposed GSAcc design. GSAcc comprises three primary modules: the Gaussian pre-processing module, which calculates transformations, projections, opacity, and color for each Gaussian; the Gaussian culling and sorting module, which performs frustum culling, computes Gaussian depths, and sorts the Gaussians based on depth; and the rasterization module, which determines the screen color for all pixels by processing the sorted Gaussians. These modules are interconnected via a bus and interface with off-chip memory. Additionally, GSAcc features a global shared buffer that caches Gaussian parameters from the compressed codebook. Both the Gaussian sorting module and the rasterization module are equipped with dedicated buffers, enhancing performance by leveraging data locality.

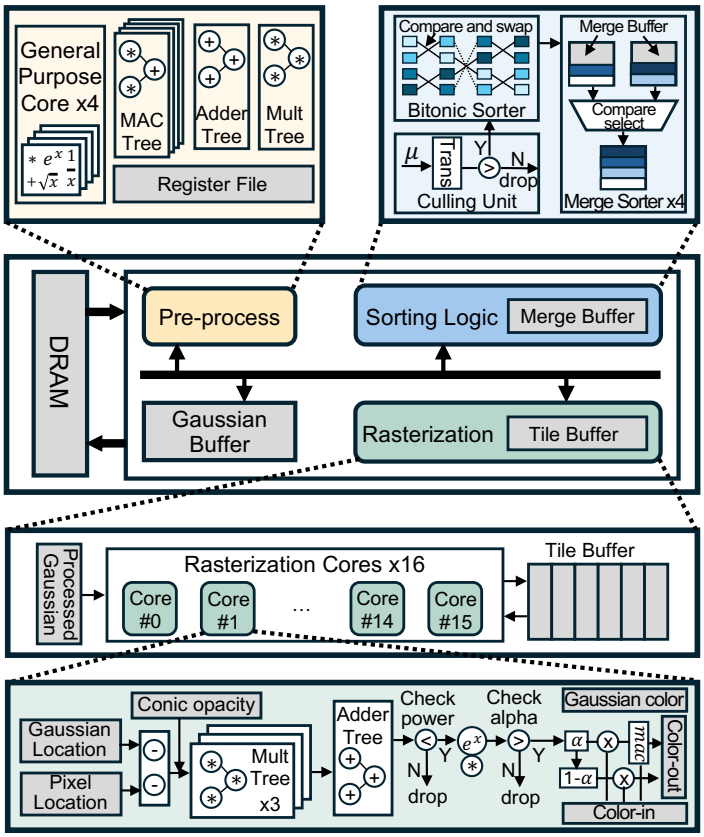


Fig. 5: Overall architecture of the proposed GSAcc. It contains a pre-processing module, a sorting module with a dedicated merge buffer, a rasterization module with a dedicated tile buffer, and a global shared Gaussian buffer.

B. Pre-processing Module

The pre-processing of a Gaussian takes the Gaussian index from the compressed codebook and camera information as input and calculates the corresponding color, conic opacity, and the tiles affected during rasterization. Since most operations involve three-dimensional points or four-dimensional quaternions, we design the pre-processing components to efficiently handle these operations. The module includes four multiply-accumulate (MAC) units, each accepts two 4-dimensional vectors and performs vector dot product. These MAC units accelerate tasks such as 3D covariance generation, view transformation, and 3D-to-2D projection, which require matrix-matrix and matrix-vector multiplications at a granularity of three or four operands per dimension. The module also features a four-input adder tree and a four-input multiplier tree to speed up continuous addition and multiplication operations in color computation from spherical harmonics. Other miscellaneous pre-processing tasks are managed by four general-purpose cores, which incorporate various arithmetic logic units, including addition, multiplication, square root, exponential functions, min/max logic, and reciprocal operations.

C. Culling and Sorting Module

The culling and sorting module of GSAcc is designed to calculate the depth of each Gaussian based on its mean location μ and the camera information represented by a view transform matrix. This module filters out Gaussians that lie outside the view frustum and sorts the remaining in-frustum Gaussians in descending order of their depth. Since in-frustum culling relies solely on the depth information of each Gaussian, the module only requires one MAC tree and one comparator to compute the z-dimension of the view transformation and perform the culling efficiently.

Sorting is a critical step in 3DGS rendering and has been a bottleneck in the original rendering pipeline. However, with the co-designed dataflow, rasterization is no longer dependent on the sorting results of the current frame, allowing us to prioritize efficiency and memory access when designing the sorting unit. We propose a two-stage accurate sorter in GSAcc. The first stage utilizes a 32-way bitonic sorter [6] that directly receives data from the culling unit. Once it collects 32 Gaussians, it begins the sort operation, comparing and swapping Gaussian pairs based on depth in a hierarchical manner. Since the compare-and-swap logic is fixed in hardware, the bitonic sorter is fully pipelined and achieves high throughput. The second stage consists of four merge sorters, each takes two sorted blocks as input and merges them into a single sorted block. This stage is efficiently implemented in hardware with compare-and-select logic, producing one sorted result per cycle per sorter. To minimize external memory accesses, the merge sorter is equipped with four dedicated merge buffers with a depth of 1024. Output chunks from the bitonic sorter are stored in two of the buffers, and the merged results remain on-chip until the buffer reaches its maximum capacity. At that point, the merged results are offloaded to DRAM, and a new merge iteration begins. Once all Gaussians are merged into large chunks matching the output buffer capacity, the external merge sorting algorithm is used for the remaining merge sorting steps.

D. Rasterization Module

The rasterization module of GSAcc is a critical component for performance, as rasterization must handle the total number of in-frustum Gaussians multiplied by the average number of pixels per Gaussian, which is typically several hundred times higher than the data processed by sorting and pre-processing modules. To maximize rasterization performance, the entire rasterization workflow is handled by a dedicated hardware block with a fully pipelined design in GSAcc. The module features 16 rasterization cores, each capable of rendering one pixel per cycle for the same Gaussian inside a tile with 16x16 pixels, achieving a maximum throughput of up to 16 pixels per cycle. Additionally, GSAcc includes one input buffer, one output buffer, and four extra stationary buffers to support tile-stationary scheduling.

Each rasterization core processes the locations, colors, and opacity of both screen pixels and the current Gaussian. The core first computes the relative Gaussian location by subtracting the pixel location from the original Gaussian location. It then generates the Gaussian power value based on the relative location and Gaussian opacity through several multiplications and additions. To enhance parallelism and reduce latency, each core is equipped with three 4-input multiplier trees and one 4-input adder tree, enabling the Gaussian power calculation to proceed in parallel. Following this, a comparator filters out invalid Gaussians by checking their power values. An exponential logic unit, combined with a multiplier, calculates the opacity contribution of the current Gaussian to the pixel. Gaussians with relatively low opacity are

discarded as they contribute minimally to the pixel color. Finally, the screen color and opacity are updated using the Gaussian color and opacity with the alpha blending algorithm.

E. Fully Parallelized Dataflow

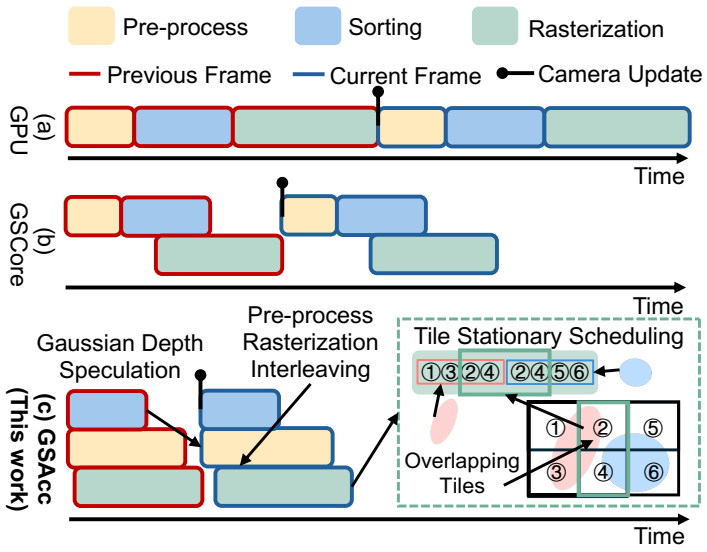


Fig. 6: Execution timeline for different dataflows in rendering for a 3DGS viewer. (a) Original GPU rendering dataflow. (b) GSCore dataflow with interleaved sorting/rasterization. (c) Proposed fully parallelized GSAcc dataflow, enabling pre-processing, sorting, and rasterization to execute concurrently.

Figure 6 illustrates the execution timeline for existing dataflows on the GPU platform [3] and the previous work GSCore, alongside our proposed dataflow for GSAcc when rendering continuous frames in a 3DGS viewer. As described in Fig. 6(a), the original rendering pipeline for a 3DGS scene on the GPU platform processes frames sequentially. Each frame undergoes pre-processing, sorting, and rasterization to produce the rendered image, with no overlap or parallelism between these steps. Fig. 6(b) shows the execution dataflow of GSCore. GSCore splits the sorting into two pipelined stages: first, it sorts all Gaussians into multiple chunks, and then it performs precise sorting within each chunk. Rasterization can start after the first chunk is precisely sorted, allowing partial overlap and parallel execution of sorting and rasterization. However, GSCore still needs to wait for the pre-processing to complete after a camera update.

Our proposed dataflow for GSAcc is illustrated in Fig. 6(c). It introduces Gaussian depth speculation by reusing the depth and order information from the previous frame as the basis for sorting in the current frame. After a camera update, GSAcc simultaneously initiates sorting and pre-processing using the updated camera information. By leveraging this depth and order speculation, Gaussians are pre-processed with depth from the last frame and view transformations and projections from the current frame. This allows rasterization to begin right after the first Gaussian is pre-processed, enabling the sorting, pre-processing, and rasterization stages to interleave and operate in parallel until all Gaussians are processed. Thus, our dataflow effectively overlaps and parallelizes these three stages, significantly enhancing rendering efficiency.

To further optimize memory access, we propose a tile-stationary aware dataflow that tracks overlapping tiles between consecutive Gaussians and adjusts the rasterization order accordingly. This ensures that overlapping tiles remain in tile buffers as much as possible during Gaussian transitions, thereby enhancing memory efficiency and reducing latency.

V. Evaluation

A. Methodology and Experiment Setup

To evaluate the performance of GSAcc, we develop an architectural simulator that models the number of cycles and off-chip memory accesses. We use the same LPDDR4-3200 as the GSCore with a maximum DRAM bandwidth of 51.2GB/s to ensure a fair comparison. Similar to GSCore, DRAM access time is simulated using Ramulator [7] and DRAM power is simulated with DRAMPower [8]. To accurately estimate on-chip power and area, we implement the RTL design in SystemVerilog. The design is synthesized with Cadence Genus using Intel16 technology under a 500MHz frequency constraint. The floating-point arithmetic units are sourced from Cadence Chipware IPs, and the buffer SRAMs are generated with an SRAM compiler from Intel. For benchmarks, we select two outdoor scenes and two indoor scenes from two datasets: "Train" and "Truck" from Tanks & Temples [9], and "Playroom" and "Drjohnson" from Deep Blending [10]. These trained models were obtained from CompGS [3], and the 3D Gaussian parameters are also quantized to FP16 for a fair comparison with GSCore.

B. Compare with Existing Design

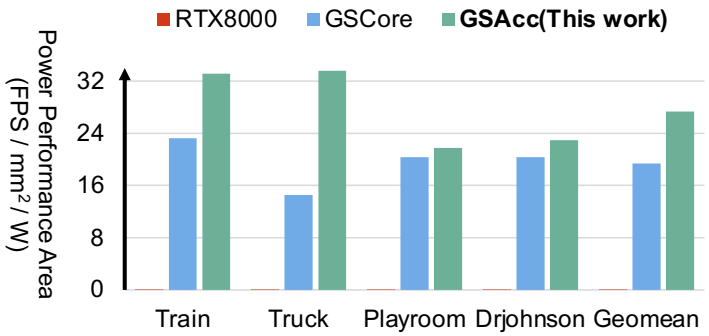


Fig. 7: Power Performance Area (PPA) comparison results of GSAcc over GPU and GSCore.

We compare our GSAcc design against both RTX8000 GPU platform and the state-of-the-art accelerator platform GSCore. To effectively evaluate the performance of different designs across varying technologies and scales, we consider Power-Performance-Area (PPA) as the primary comparison metric. Consequently, we utilize rendering frames per second per square millimeter per watt (FPS/mm²/W) as our key metric. Fig. 7 illustrates the PPA results achieved by GSAcc. Results for the GPU platform are profiled with a code base from Compact3D and the results for GSCore are directly referenced from the original paper. When comparing to the GPU platform, GSAcc achieves up to 1.66×10^4x higher PPA and an average improvement of 1.35×10^4x. Additionally, GSAcc outperforms the state-of-the-art accelerator GSCore by achieving up to 2.30x higher PPA and an average of 1.41x higher PPA.

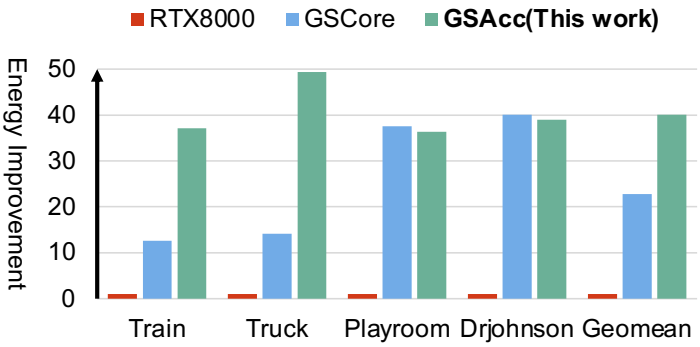


Fig. 8: Energy improvement results of GSAcc over GPU and GSCore.

In terms of energy efficiency improvement, we calculate the energy consumed to render a single frame and then determine the factor by which energy usage is reduced compared to the GPU platform. Figure 8 presents the comparison results for GSAcc. GSAcc can save up to 48.7x energy per frame compared to the GPU platform and up to 2.9x energy per frame compared to GSCore. On average, GSAcc achieves a 39.8x improvement in energy efficiency over the GPU platform and a 1.74x improvement compared to GSCore.

C. Performance Improvement Analysis

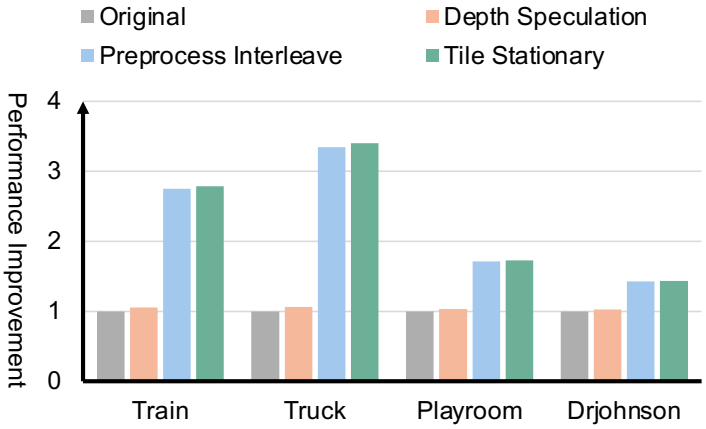


Fig. 9: Breakdown of GSAcc performance improvement with each optimization technique.

To further investigate the contribution of each optimization method applied in GSAcc, we first ran the original dataflow on the GSAcc architecture to establish a performance baseline, as shown in Fig. 9. We then introduce Gaussian depth speculation, enabling parallel execution of the sorting process. This enhancement results in a performance improvement of 4%–5%, primarily because the sorter module in GSAcc is optimized. Next, by incorporating pre-processing overlapping, we observe a significant performance increase across all scenes, as pre-processing and rasterization could operate concurrently without storing intermediate results. Finally, implementing tile stationary adds an additional 5% performance boost and reduces memory access energy by decreasing off-chip memory accesses. These incremental optimizations collectively demonstrate the effectiveness of our design choices in enhancing GSAcc's overall performance and energy efficiency.

D. Area/Power Breakdown

TABLE I: Synthesized area and power of GSAcc submodules

Module	Area (mm^2)	Power (W)
Pre-processing module	0.0507	0.0297
Culling and sorting module	0.1006	0.1206
Rasterization module	0.1377	0.2395
SRAMs + Others	0.1023	0.0187
Total	**0.3913**	**0.4085**

Table I presents the synthesized area and power results for GSAcc. Our GSAcc architecture consumes a total of 0.4085W and occupies an area of 0.3913mm². The rasterization module is the primary contributor to both area and power consumption, highlighting its role as the key high-performance component. Additionally, the sorter module, particularly the bitonic sorter, significantly impacts both power and area. In contrast, SRAM buffers occupy a large portion of the area but contribute minimally to power consumption.

VI. Related Work

Given the exceptional performance of 3DGS and its promising potential in AR/VR applications, efforts to accelerate 3DGS have captured considerable attention within the algorithmic community. Several studies have demonstrated the feasibility of compressing Gaussian models to reduce their size [11]–[13], thereby enhancing rendering speed on GPUs by lowering memory bandwidth requirements. Additionally, techniques such as Gaussian opacity regulation [14] have been developed to accelerate rendering by decreasing the number of Gaussians during model training, further improving efficiency.

On the system and architecture front, researchers have begun exploring lower-level acceleration opportunities for 3DGS. For instance, FlashGS [15] provides an optimized CUDA library specifically designed for accelerating 3DGS rendering on GPU platforms, while GSCore [4] introduces a dedicated hardware accelerator tailored for this purpose. However, these approaches have not yet achieved a fully parallelized rendering pipeline. Moreover, they have not investigated the utilization of information from previous frames to enhance rendering performance in real-time AR/VR applications.

VII. Conclusion

This paper proposed GSAcc, an architecture/dataflow co-design solution to accelerate 3D Gaussian Splatting. GSAcc effectively leverages frame-to-frame information reuse and optimizes the rendering pipeline through parallelized sorting and rasterization, significantly enhancing performance and energy efficiency compared to existing GPU and accelerator platforms. Our comprehensive evaluations demonstrate that GSAcc delivers substantial improvements in PPA and energy efficiency, making it a viable solution for immersive and resource-constrained environments.

Acknowledgment

We thank Intel for PDK and memory compiler support, Cadence for IP and EDA tools support, and the UT iMAGINE consortium for supporting this research project.

REFERENCES

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: representing scenes as neural radiance fields for view synthesis," *Commun. ACM*, vol. 65, no. 1, p. 99–106, Dec. 2021. [Online]. Available: https://doi.org/10.1145/3503250

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[3] K. Navaneet, K. P. Meibodi, S. A. Koohpayegani, and H. Pirsiavash, "Compgs: Smaller and faster gaussian splatting with vector quantization," 2024. [Online]. Available: https://arxiv.org/abs/2311.18159

[4] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim, "Gscore: Efficient radiance field rendering via architectural support for 3d gaussian splatting," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 497–511. [Online]. Available: https://doi.org/10.1145/3620666.3651385

[5] B. A. Wallace, "Merging and transformation of raster images for cartoon animation," *SIGGRAPH Comput. Graph.*, vol. 15, no. 3, p. 253–262, Aug. 1981. [Online]. Available: https://doi.org/10.1145/965161.806813

[6] T. Nakatani, S.-T. Huang, B. Arden, and S. Tripathi, "K-way bitonic sort," *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 283–288, 1989.

[7] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[8] K. Chandrasekar, C. Weis, Y. Li, S. Goossens, M. Jung, O. Naji, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," http://www.drampower.info.

[9] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: benchmarking large-scale scene reconstruction," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017. [Online]. Available: https://doi.org/10.1145/3072959.3073599

[10] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Trans. Graph.*, vol. 37, no. 6, Dec. 2018. [Online]. Available: https://doi.org/10.1145/3272127.3275084

[11] S. Niedermayr, J. Stumpfegger, and R. Westermann, "Compressed 3d gaussian splatting for accelerated novel view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024, pp. 10 349–10 358.

[12] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps," 2024. [Online]. Available: https://arxiv.org/abs/2311.17245

[13] M. T. Bagdasarian, P. Knoll, Y.-H. Li, F. Barthel, A. Hilsmann, P. Eisert, and W. Morgenstern, "3dgs.zip: A survey on 3d gaussian splatting compression methods," 2024. [Online]. Available: https://arxiv.org/abs/2407.09510

[14] A. Guédon and V. Lepetit, "Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering," *CVPR*, 2024.

[15] G. Feng, S. Chen, R. Fu, Z. Liao, Y. Wang, T. Liu, Z. Pei, H. Li, X. Zhang, and B. Dai, "Flashgs: Efficient 3d gaussian splatting for large-scale and high-resolution rendering," 2024. [Online]. Available: https://arxiv.org/abs/2408.07967

Identifying System-on-Chip Security Assets with Structure-Based Analysis

Wei-Kai Liu[§], Benjamin Tan[†], and Krishnendu Chakrabarty[‡]
[§]Duke University, [†]University of Calgary, [‡]Arizona State University
weikai.liu@duke.edu, benjamin.tan1@ucalgary.ca, krishnendu.chakrabarty@asu.edu

***Abstract*—In the evolving field of hardware design, ensuring the security of System-on-Chips (SoCs) has become increasingly vital. As SoCs grow in complexity, integrating components from various sources, the identification and protection of security assets are crucial to prevent vulnerabilities. Traditional methods of identifying these assets are manual and time-intensive. To address this challenge, automated tools for security asset identification are essential, enabling faster and more accurate detection of critical assets early in the design process. In this paper, we propose a framework for the automated identification of security assets within SoCs. By transforming register-transfer level (RTL) code into graphs and leveraging deep neural networks (DNNs) to classify assets based on their structural patterns, our approach can effectively differentiate between security and non-security assets. Experimental results show that the proposed method achieves high classification accuracy, with the model reaching up to 99% accuracy in identifying security assets, significantly reducing the need for manual intervention.**

I. Introduction

System-on-chips (SoCs) have become vital to modern technology, driving innovation in industries ranging from consumer electronics to healthcare, mobile devices, and autonomous vehicles [1]–[4]. As these systems increasingly manage sensitive data and perform critical operations, ensuring their security has become a top priority for designers and system integrators [5]–[8]. Key security elements within SoCs, such as encryption keys, configuration data, and access control signals, must be rigorously protected to maintain the system's integrity and confidentiality [9].

As SoCs become more complex, integrating a diverse array of IPs from various sources, the potential for security vulnerabilities increases [10]–[13]. Furthermore, the interactions between these different components can lead to unforeseen security issues that malicious users could exploit after deployment. Common Weakness Enumerations (CWEs) proposed by MITRE highlight various examples of prevalent security bugs and the corresponding exploits observed in the industry [14]. While security assets are typically integrated with various protection mechanisms to ensure the safety of each IP, these mechanisms can be improperly implemented, leaving assets vulnerable to exploitation [15]. As highlighted by [14], many security assets are inadequately protected due to incorrect access control policies or misconfigurations. Although some hardware patching techniques have been proposed to protect security assets during runtime [5], [16], they cannot guarantee complete protection. Therefore, system integrators must thoroughly review all security assets across IPs and their associated protection mechanisms to ensure the SoC's security.

To address these challenges, it is crucial to prioritize security from the earliest stages of SoC design [17]. This involves identifying the most critical assets that require protection and ensuring their security throughout the design process. Several approaches have been proposed to safeguard assets in SoCs, such as incorporating security policies or properties alongside simulations to verify the safety of designs [18]–[21]. However, these methods often require the manual definition and inspection of security assets and their corresponding protection mechanisms, relying heavily on the designer's efforts. Previous works [22], [23] identify assets in a design by tracing the data flow from a list of primary assets provided by IP vendors. However, this approach relies heavily on designers, so it may overlook other security assets if the primary assets are not explicitly defined.

To enable pre-silicon security analysis, we propose a method for identifying security assets in SoCs at the register-transfer level (RTL) automatically. Therefore, system integrators can easily gather security assets of different IPs without manually delving into each design. Our approach captures the data flow and control flow of each signal, converting them into numerical vectors that serve as signatures for identifying other assets with similar structures. Since assets of the same type usually have the same protection mechanism, i.e., the data/control flow is similar, this approach can effectively identify security assets by clustering signals with similar structures. By minimizing the manual workload and reducing the amount of user-provided information, this method aims to streamline the early stages of security analysis, enhancing both its robustness and accuracy.

Our main contributions are as follows:

- We introduced a framework to automate the identification of security assets in SoCs by analyzing structural patterns of RTL signals, reducing the need for human efforts.
- We developed a structure-based approach that captures both control flow and data flow information from RTL, facilitating the identification of security assets through graph analysis.
- We employed a DNN model to classify security and non-security assets and achieve high accuracy across multiple SoC platforms.

The remainder of the paper is organized as follows: Section II provides an overview of asset definitions and security counter-

*This work was supported in part by the Semiconductor Research Corporation (SRC) under Tasks 3065.001, 3106.001, and 3198.001, and by a gift from Intel Corporation.

measures, while Section III reviews relevant prior work. The proposed framework is introduced in Section IV, followed by experimental results in Section V. Finally, the paper concludes in Section VI.

II. Background

A. Definition of Security Assets

Accellera's SA-EDI standard [24] defines assets as "anything of value or importance that is used, produced, or protected within the IP." OpenTitan, the largest commercial-scale open-source hardware root-of-trust integrating various security IPs, has documented all asset types in their projects, with a selection shown in Table I. However, for SoC security, we focus only on assets that impact information security, specifically those that affect confidentiality, integrity, and availability, as represented by the CIA triad. These are referred to as security assets.

In previous works [18], [23], security assets are primarily classified into two: primary assets and secondary assets.

- **Primary Assets**: Critical design components essential to the security of a SoC. Their integrity is vital, and their compromise can lead to significant security breaches.
- **Secondary Assets**: Supportive components linked to primary assets. While not directly critical, their compromise can weaken the overall security of the SoC.

For example, keys or passwords are classified as primary assets, while data registers used to transmit primary assets are considered secondary assets. Although data registers may not typically contain sensitive information, they can still be exploited by attackers to access primary assets if left unprotected. Similarly, as shown in Table I, some asset types, such as RST and CTR, are not usually security-relevant but can become secondary assets in specific scenarios. As such, there is no definitive standard for classifying security assets. Therefore, this paper proposes an approach that evaluates each signal's data and control flow complexity, as well as its relationship with other signals, to identify security assets, rather than relying solely on asset type classifications.

TABLE I: Common asset types proposed by [25].

Asset Name	Meaning
KEY	A key (secret data)
CTRL	Logic used to steer hardware behavior
ADDR	An address
DATA_REG	A configuration data register
CTR	A counter
RST	A reset signal
MEM	A generic data memory
CONFIG	A configuration register
BUS	Data transferred on a bus
CONSTANTS	A netlist constant

B. Security Countermeasures

Security countermeasures are protection mechanisms designed to safeguard security assets. For security-critical IPs, these countermeasures are commonly implemented to prevent attacks or side-channel analysis. The specific countermeasures employed vary depending on the type of asset. Some common security countermeasures used by OpenTitan are listed in Table II.

Since assets of the same type typically share the same countermeasures, especially when they originate from the same design house, these countermeasures can be leveraged for identifying security assets. Such countermeasures introduce additional data or control flows to the assets to provide protection. For example, **REGWEN** and **REGREN** condition asset access on the *wen* and *ren* signals, ensuring that assets can only be accessed when these enable signals are pulled high. Usually, all the *wen* and *ren* signals are typically generated in similar ways or through modularized logic, with differences only in their switching conditions. Thus, if one security countermeasure can be identified automatically, all assets using the same protection mechanism can be identified accordingly. Since the number of security countermeasures is typically much smaller than the number of security assets, designers can significantly reduce effort by providing a list of countermeasures instead of security assets.

TABLE II: Common countermeasures used in OpenTitan [25].

Countermeasure Name	Meaning	Associated Assets
REGWEN	A register write enable is used to protect the asset from write access	ADDR, CTRL
SHADOW	The asset has a shadow replica to cross-check against	CONFIG
INTEGRITY	The asset has integrity protection from a computed value	CONFIG, MEM
REGREN	A register read enable is used to protect the asset from read access	CONFIG, MEM

C. Hyperflow Graph

A hyperflow graph (HFG) is a specialized intermediate representation designed to support and enhance security analysis and visualization in hardware systems [26]. It maps out the intricate connections between different design variables, capturing both direct and conditional data flows. This graph not only helps in tracing how information moves through the system but also provides crucial insights into security-relevant aspects such as key assets, boundary definitions, and the execution of specific functions during simulation. In the proposed work, we simplified the original HFG proposed in [26] to represent the relationship among signals.

In this HFG graph, each node represents a signal, or a single bit of a signal if each bit has different control or data flows. An edge exists between two nodes whenever there are any control or data flows between them. A solid edge represents a direct assignment between two signals, while a dotted edge is used for a conditional assignment. For example, in the following RTL statement:

```
if (ctrlA or ctrlB)
  dataB <= dataA;
else
  dataB <= dataC;
```

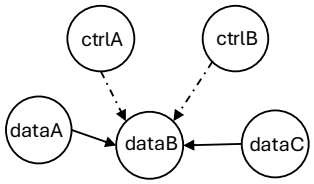


Fig. 1: An example hyperflow graph.

two solid edges pointing to *dataB* from *dataA* and *dataC* and two dotted edges pointing to *dataB* from *ctrlA* and *ctrlB* will be created.

An example hyperflow graph is shown in Figure 1. Since the value of *dataB* is determined by either *dataA* or *dataC*, these direct assignments are represented by solid edges. The signals *ctrlA* and *ctrlB*, which control the value of *dataB*, are connected with dotted edges to indicate control flow. In simpler terms, signals on the right-hand side (RHS) will have solid edges pointing to the signals on the left-hand side (LHS), while signals involved in logical statements, such as if and case statements, will be connected with dotted edges to the signals on the LHS within their respective scopes.

The HFG effectively clarifies the relationships between signals by stripping away unnecessary information from the RTL and focusing solely on the data and control flows among signals. By tracing the nodes through different types of edges—solid for direct assignments and dotted for control flows—we can identify distinct structural patterns associated with various assets, as each asset type exhibits unique characteristics.

III. Related Prior Work

Several approaches have been proposed to protect security assets in SoCs, but most rely heavily on manual identification or predefined asset sets. For example, Meza et al. [26] used information flow tracking with hyperflow graphs to verify the security of the OpenTitan hardware root of trust, focusing on OpenTitan's security assets. However, their method required manual inspection and did not automate asset identification.

Dipu et al. [27] introduced a framework to generate system architectures and comprehensive security policies tailored to security assets, considering various vulnerabilities. While their approach is policy-driven, it requires the security assets to be defined in advance. Similarly, design-for-test (DFT) logic, which enhances the observability and controllability of security assets for testing purposes, must be reviewed carefully. As noted in [28], if not properly secured, DFT logic can be exploited by attackers to gain access to these assets. Their study also relied on manually selecting security assets for analysis.

[9] proposed an automated approach for scanning SoC designs to identify potential security vulnerabilities based on the CWE database [14]. However, their method lacks generalizability, as it only implements algorithms for a limited set of specific weaknesses. Additionally, each scanner is tailored to a particular CWE, which means that significant designer effort would be required to develop a comprehensive set of algorithms capable of discovering a broader range of security bugs in SoCs.

[18] proposed an algorithm to generate security-critical properties for SoCs by identifying vulnerabilities and creating corresponding protective properties for selected assets. Ray et al. [20] and Hu et al. [19] evaluate the security properties of hardware components systematically, identifying potential vulnerabilities, and generating policies to protect IP assets during verification. Despite these advancements, all of these methods still depend on manual asset identification or predefined asset sets, limiting their scalability and automation potential.

[23] is the only work we found that identifies security assets automatically. By defining a set of primary assets, it employs algorithms to track information flow and subsequently identify secondary assets. However, this approach is limited to identifying only those secondary assets that have data or control flow to or from the predefined primary assets, which must be provided in advance. As a result, if IP vendors fail to list all the primary assets, some security assets may be overlooked. To address this limitation, the proposed work aims to automatically identify both primary and secondary assets without requiring the set of primary assets from designers.

IV. Proposed Framework

The proposed framework is shown in Figure 2. The process begins by collecting the source code and design packages from a given SoC. We then parse the Verilog/SystemVerilog design code and generate abstract syntax trees (ASTs) using Verific's parser [29]. Each AST corresponds to a module, capturing all the essential information from the RTL code. To extract the necessary details for building the HFG, we implement various visitors to traverse these ASTs.

A. Building HFG

To construct the HFG, we primarily need to capture the control flow and data flow between signals. To achieve this, we design visitors that detect all conditional blocks, such as "case" and "if" statements, as well as conditional assignments, where control signals influencing the values of other signals are identified. Any signals appearing in the sensitivity list are recognized as control signals, regardless of the boolean operators used between them. In addition, both blocking and non-blocking assignments are identified, and the signals on the LHS and RHS are mapped to establish the corresponding data flow. Similarly, when multiple signals appear on the RHS (e.g., $C = A\&B$), the data flow is noted as originating from both A and B towards C. To trace signals across hierarchical modules, we record the port names of all submodules and their corresponding signal names. Once all modules have been visited, a bottom-up traversal is performed to fill in any missing connections where signals pass through submodules. During this traversal, missing submodules' subgraphs are created and linked to the corresponding module. We implement a tool in python to plot the HFG to ensure the correctness of our implementation. A visualization of the HFG for *opt_ctrl.sv* is

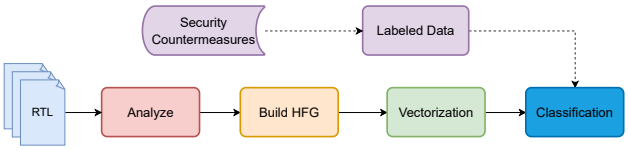


Fig. 2: The proposed framework for security asset identification.

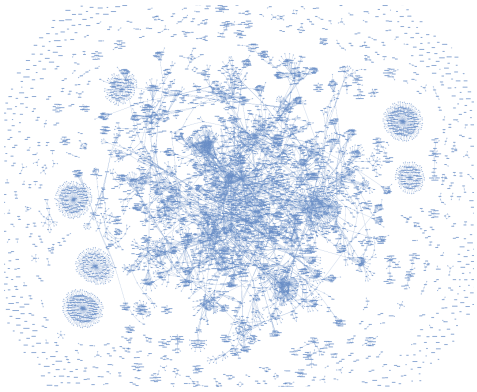


Fig. 3: HFG visualization of the OTP controller in OpenTitan. A multi-bit signal (e.g., a bus) is represented as either a single node or multiple nodes, where each node corresponds to a bit with distinct data or control flows. When output signals in the top module lack data/control flows into submodules, cluster-type sub-graphs are generated.

presented in Figure 3, allowing for easy tracing of signals' data/control flows and providing a clear understanding of the relationships among signals.

B. Vectorization

To perform security asset identification, we extract the structural information from the HFG of each signal to represent its connections. Since the HFG is a directed graph with two types of connections, representing data and control flows respectively, each node can have up to four distinct types of neighbors: conditional predecessors (CP), direct predecessors (DP), conditional successors (CS), and direct successors (DS). Predecessors are nodes that have edges pointing to the target node, while successors are nodes that the target node points to. Direct connections represent data flow, and conditional connections represent control flow, visualized as solid and dotted lines, respectively, as shown in Figure 1.

The number of these four different types of neighbors provides insight into a signal's characteristics. For example, reset, enable, or other control signals that frequently appear in 'if' or 'case' statements will have many conditional successors and few, if any, direct successors. Conversely, data registers or signals carrying memory values, which are often accessed or passed with other values, will have many direct successors and predecessors. By analyzing each signal based on these characteristics, we can effectively categorize signals into distinct groups. These four numbers can then be combined into a vector, serving as a unique signature for each signal.

However, some signals may appear similar based on these four numbers but actually have significantly different connections beyond the first layer of neighbors. To address this issue, we incorporate additional information from the HFG by capturing two layers of neighbors for each signal. In implementation, we achieve this by combining the vectors of all neighboring nodes of the target node. To prevent information loss during the combination process, we group neighbors by type (CP, DP, CS, DS) and calculate the average vector for each group. Therefore, each vector can represent the general characteristics for each type of neighbors of the target node, while the original 4-dimensional vector still specifies the number of each type of neighbors. As a result, in addition to the original 4-dimensional vector, we generate four additional 4-dimensional vectors corresponding to each type of connection for the target node. Ultimately, this results in a 20-dimensional vector representing a signal, which is then used for classification.

C. Classification

Once each signal is vectorized, these vectors serve as unique signatures for the signals, allowing for classification. We employ a Deep Neural Network (DNN) to train a model capable of identifying security assets. To train the model, labeled datasets are required, which must be provided by the designers. However, designers only need to supply a small dataset representing each type of security asset, as different asset types exhibit distinct security countermeasures. These countermeasures result in varied connections within the HFG, and the model is trained to recognize and classify these differences effectively.

By identifying the structure of each protection mechanism, this approach is agnostic to the varying keywords or naming conventions used by different IP vendors. Additionally, because we consider not only the closest neighbors of the target node (i.e., those with a distance of 1), the method is more robust against small design modifications. For example, even if two assets are protected using the same mechanism, the number of CP, DP, CS, and DS connections may differ slightly. In such cases, the DNN can still classify them into the same category, as long as the Euclidean distance between the signals is within a threshold and their signature vectors exhibit similar characteristics.

Although many Graph Neural Network (GNN) approaches have demonstrated strong performance [30], [31], graph transferability remains a significant challenge [32]. Graph transferability refers to the problem where a GNN model, trained on one graph, struggles to generalize to another, resulting in poor performance on unseen graphs. However, our objective is to identify specific structural patterns within security assets. Therefore, regardless of how the HFG is constructed, the known assets must still be accurately identified. Since HFGs can vary significantly between different SoCs or across different hierarchical levels within the same SoC, we chose not to use GNNs for classification. Moreover, as each dimension of the signal vector contains all the necessary information, a

DNN is much more energy- and time-efficient compared to a GNN.

V. Experimental Results

A. Experimental Setup and DNN Hyperparameters

Our experiments are conducted on a machine equipped with an NVIDIA TITAN V. The training hyperparameters are as follows: the model is trained for 100 epochs using the Adam optimizer with a learning rate of 10^{-3}, which decayed by a factor of 0.1 every 5 epochs. A batch size of 32 was used during training. The DNN model consists of four fully connected layers, each followed by batch normalization and ReLU activation. A dropout rate of 0.3 was applied after each fully connected layer to reduce overfitting by randomly disabling neurons during training. The number of neurons in the layers are 128, 64, 16, and a task-specific output size. Input normalization was applied to ensure stable and efficient training. The cross-entropy loss function was employed for classification tasks, with softmax applied at the output layer. We utilized 10-fold cross-validation to ensure model robustness and generalization. The primary evaluation metric used was accuracy, measured on the validation set during each fold.

B. Overview of SoCs

We conduct the experiments on OpenTitan and Openpiton (from Hack@Dac21 [33]). The number of scanned modules by Verific, along with the number of nodes and edges in the HFGs, and other information of these SoCs, are presented in Table III. Due to the absence of labeled data from these SoCs, we manually reviewed and selected security assets based on our understanding of the system. OpenTitan, designed with security as a primary focus, has more structured countermeasures, security features, and clearer asset identifiers. A block diagram of the OpenTitan Earl Grey chip is shown in Figure 5, showcasing its multiple security peripherals within the system.

For OpenTitan, we selected assets in alignment with the proposed countermeasures, as shown in Table II. We identified security assets by searching for specific keywords associated with the countermeasures, including “regwen”, “gated_we”, “shadowed”, and “intg_err”. In contrast, OpenPiton is more general-purpose, requiring more generalized keywords for asset classification. For Openpiton, we use “mem”, “en”, “_ctrl”, “key” to search for security assets, while keywords such as “rdata”, “wdata”, and “address” to search for assets that may not be security-relevant. Signals that did not match any of the specified keywords were classified as non-security assets.

The runtime for completing the entire framework is presented in Table III. In contrast to manual inspection—which typically takes several minutes to review an RTL file with 100 lines, and requires a deep understanding of each IP to accurately identify security assets—our automated approach significantly reduces the time required. Specifically, our method identifies security assets from over 300 modules, comprising nearly 240k lines of code, in just 13 seconds.

TABLE III: Overview of SoC metrics used in the experiment, including the number of scanned modules, lines of code parsed, hyperflow graph details, and the corresponding runtime.

SoC Name	# of Modules	LoC	# of Nodes	# of Edges	Time (s)
OpenTitan	332	239790	1455692	2375044	13.6
Hack@DAC21 (Openpiton)	128	166396	1397004	3076387	11.5

C. Results Analysis

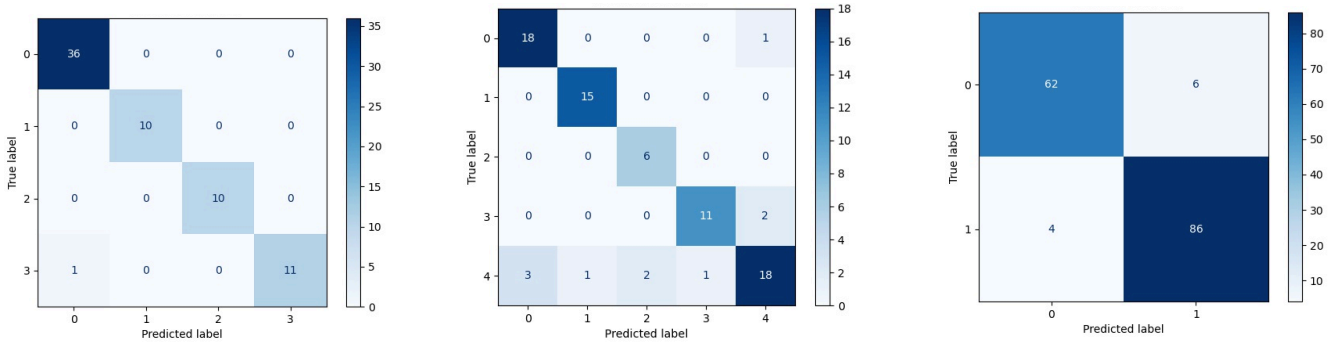
In OpenTitan, we used four different keywords to classify signals into distinct asset categories, i.e., “regwen”, “gated_we”, “shadowed”, and “intg_err”. Initially, we collected the signals associated with each keyword and ran the classification model. The resulting classification achieved an accuracy of 99%, with the confusion matrix depicted in Figure 4 (a). This high accuracy demonstrates the model’s ability to distinguish between the selected asset types when only security-relevant signals are present.

Next, we incorporated non-security assets into the dataset. As shown in Figure 4 (b), the classification accuracy dropped to 92%. A closer look at the confusion matrix reveals that many non-security assets (label 4) were incorrectly classified as different types of security assets. This misclassification is likely due to the limitations of our keyword-based labeling system. While the keywords effectively capture many security assets, there are other security-relevant signals that do not match these keywords but share similar protection mechanisms. As a result, the model misclassifies some non-security assets that exhibit similar patterns.

Lastly, we combined the four asset types into a single category, labeling all of them as “security assets,” and performed a binary classification against the non-security assets. The result, as shown in Figure 4 (c), achieved a 94% accuracy. This binary classification simplified the task, but still highlights the challenges of accurately identifying security signals.

We then performed the same three classification tasks for OpenPiton. The results yielded accuracies of 93%, 86%, and 88%, following a similar trend as observed with OpenTitan. When incorporating different classes of security assets with non-security assets, the accuracy is the lowest for both SoCs. This is likely due to some security assets being mislabeled as non-security assets during the keyword-searching process, which, although effective, is not perfect.

However, when the various categories of security assets are grouped together for classification, we achieve the highest accuracy. This indicates that the different types of security assets exhibit distinct structural patterns in their vector representations, allowing for easy differentiation. Notably, these assets come from different IPs, which means their connections may not be identical, yet they are still correctly classified under the same category. Even non-security relevant assets in OpenPiton, such as “rdata”, “wdata”, and “address”, can be accurately identified. This demonstrates that the DNN model is effective at distinguishing signals based on their structural patterns, as long as these patterns differ sufficiently.



(a) (b) (c)

Fig. 4: Confusion matrices for security asset classification. (a) shows the classification results for four distinct types of security assets in OpenTitan, based on the keywords: regwen, gated_we, shadowed, and intg_err. (b) presents the classification results for both the four security asset types (labels 0-3) and non-security assets (label 4). (c) displays the binary classification results, differentiating between security assets (label 0, matching the four keywords) and non-security assets (label 1, signals not matching any keyword).

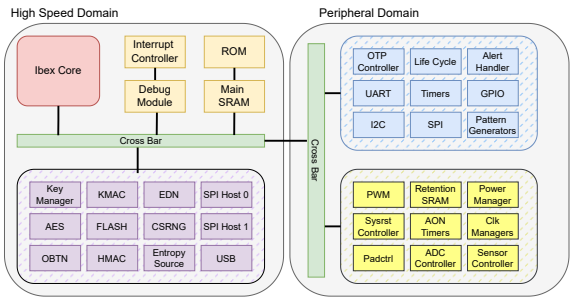


Fig. 5: Block Diagram of the OpenTitan Earl Grey Chip [25].

To further understand the importance of each input vector dimension, we plotted the absolute values of the first layer's model weights, as shown in Figure 6. The 20 dimensions consist of the following:

- 4 dimensions represent the number of each type of neighbor (CP, DP, CS, DS) for the target node.
- 16 dimensions are the averaged characteristics of neighbors in each of the four groups (CP, DP, CS, DS), with 4 dimensions per group (e.g., CP-CP, CP-DP, ..., DS-DS), representing the interactions between different neighbor types.

The weight distribution appears relatively uniform, suggesting that each dimension of the signature vector contributes similarly to the classification decision. This uniform distribution highlights that the DNN is leveraging all available features without relying heavily on any single dimension, which indicates a balanced representation of the input data.

Such uniformity in the weight distribution also suggests that the extracted features are comprehensive and non-redundant, meaning that no single feature dominates the decision-making process. This is particularly important as security assets often display subtle variations that require a broad set of features for accurate identification. This distribution implies that the DNN is learning to generalize well across various protection mechanisms, using multiple dimensions to capture different aspects of the signals.

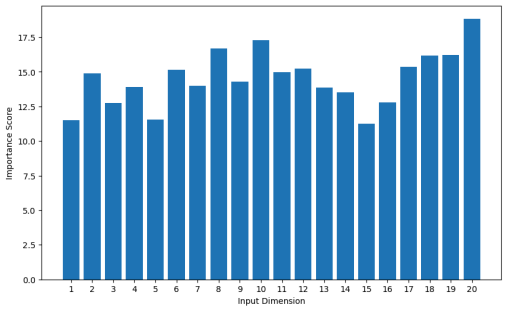


Fig. 6: The importance of each input dimension based on first layer weight.

VI. Conclusion

In this paper, we introduced a framework for the automated identification of security assets in SoCs. By constructing hyperflow graphs (HFGs) from RTL code and employing deep neural networks (DNNs) for classification, we were able to accurately identify security-relevant signals. Our experiments showed that the model effectively distinguished between security and non-security assets, even in the presence of architectural complexities and limitations associated with keyword-based labeling. By leveraging structural patterns within the signals, the proposed method significantly reduces manual intervention, streamlining and improving the efficiency of the security analysis process. The model achieved classification accuracies of 99% and 93% across different asset types for two distinct SoCs, demonstrating its robustness and adaptability.

REFERENCES

[1] K. Tsantikidou and N. Sklavos, "Flexible security and privacy, system architecture for IoT, in healthcare," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2022.

[2] A. Ishtiaq, M. U. Khan, S. Ali, K. Habib, S. Samer, and E. Hafeez, "A review of system on chip (SoC) applications in internet of things (IoT) and medical," in *International Conference on Advances in Mechanical Engineering*, 2021.

[3] D. Ariando, C. Chen, M. Greer, and S. Mandal, "An autonomous, highly portable NMR spectrometer based on a low-cost System-on-Chip (SoC)," *Journal of Magnetic Resonance*, 2019.

[4] S. R. Sridhara *et al.*, "Microwatt embedded processor platform for medical system-on-chip applications," *IEEE Journal of Solid-State Circuits*, 2011.

[5] W.-K. Liu, B. Tan, J. M. Fung, R. Karri, and K. Chakrabarty, "Hardware-supported patching of security bugs in hardware IP blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[6] J. Abella *et al.*, "Security, reliability and test aspects of the risc-v ecosystem," in *IEEE European Test Symposium (ETS)*, 2021.

[7] F. Farahmandi, A. Srivastava, G. Di Natale, and M. Tehranipoor, "Introduction to the special issue on CAD for security: Pre-silicon security sign-off solutions through design cycle," 2023.

[8] M. Akyash and H. M Kamali, "Evolutionary large language models for hardware security: A comparative survey," in *Proceedings of the Great Lakes Symposium on VLSI*, 2024.

[9] B. Ahmad *et al.*, "Don't cweat it: Toward cwe analysis techniques in early stages of hardware design," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2022.

[10] B. Ahmed *et al.*, "Quantifiable assurance: from IPs to platforms," *arXiv preprint arXiv:2204.07909*, 2022.

[11] J. Knechtel *et al.*, "Towards secure composition of integrated circuits and electronic systems: On the role of EDA," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[12] Z. Wang, L. Alrahis, L. Mankali, J. Knechtel, and O. Sinanoglu, "Llms and the future of chip design: Unveiling security risks and building trust," *arXiv preprint arXiv:2405.07061*, 2024.

[13] H. Pearce, R. Karri, and B. Tan, "High-level approaches to hardware security: A tutorial," *ACM Transactions on Embedded Computing Systems*, 2023.

[14] S. Christey, J. Kenderdine, J. Mazella, and B. Miles, "Common weakness enumeration," *Mitre Corporation*, 2013.

[15] G. Dessouky, D. Gens, P. Haney, G. Persyn, A. Kanuparthi, H. Khattri, J. M. Fung, A.-R. Sadeghi, and J. Rajendran, "Hardfails: insights into software-exploitable hardware bugs," in *USENIX Security Symposium*, 2019.

[16] W.-K. Liu, B. Tan, J. M. Fung, and K. Chakrabarty, "Theoretical patchability quantification for IP-level hardware patching designs," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024.

[17] S. R. Rajendran, N. F. Dipu, S. Tarek, H. M. Kamali, F. Farahmandi, and M. Tehranipoor, "Exploring the abyss? unveiling systems-on-chip hardware vulnerabilities beneath software," *IEEE Transactions on Information Forensics and Security*, 2024.

[18] N. Farzana, F. Rahman, M. Tehranipoor, and F. Farahmandi, "SoC security verification using property checking," in *IEEE International Test Conference (ITC)*, 2019.

[19] W. Hu, A. Althoff, A. Ardeshiricham, and R. Kastner, "Towards property driven hardware security," in *International Workshop on Microprocessor and SOC Test and Verification (MTV)*, 2016.

[20] S. Ray and Y. Jin, "Security policy enforcement in modern SoC designs," in *International Conference on Computer-Aided Design (ICCAD)*, 2015.

[21] S. Ray, E. Peeters, M. M. Tehranipoor, and S. Bhunia, "System-on-chip platform security assurance: Architecture and validation," *Proceedings of the IEEE*, 2017.

[22] N. Farzana, A. Ayalasomayajula, F. Rahman, F. Farahmandi, and M. Tehranipoor, "Saif: Automated asset identification for security verification at the register transfer level," in *IEEE VLSI Test Symposium (VTS)*, 2021.

[23] A. Ayalasomayajula, N. Farzana, M. Tehranipoor, and F. Farahmandi, "Automatic asset identification for assertion-based SoC security verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[24] Accellera Systems Initiative, "Security annotation for electronic design integration standard," 2021.

[25] "Opentitan," *https://opentitan.org/*.

[26] A. Meza and R. Kastner, "Information flow coverage metrics for hardware security verification," *arXiv preprint arXiv:2304.08263*, 2023.

[27] N. F. Dipu, F. Farahmandi, and M. Tehranipoor, "SoC security properties and rules," *Cryptology ePrint Archive*, 2021.

[28] G. K. Contreras, A. Nahiyan, S. Bhunia, D. Forte, and M. Tehranipoor, "Security vulnerability analysis of design-for-test exploits for asset protection in SoCs," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

[29] "Verific design automation," *http://www.verific.com*.

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

[31] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[32] L. Ruiz, F. Gama, and A. Ribeiro, "Graph neural networks: Architectures, stability, and transferability," *Proceedings of the IEEE*, 2021.

[33] HACK@EVENT, "HACK@DAC21," 2022. [Online]. Available: https://hackatevent.org/hackdac21/

An Efficient Bit-level Sparse MAC-accelerated Architecture with SW/HW Co-design on FPGA

Chenming Zhang, Lei Gong*, Chao Wang* and Xuehai Zhou
School of Computer Science and Technology, Suzhou Institute for Advanced Research
University of Science and Technology of China

***Abstract*—Exploring bit-level sparsity in the MAC process has been proven to be an important method for improving the efficiency of neural network feedforward processing. The reconfigurable platform offers possibilities for identifying the bit-level unstructured redundancy during inference with different DNN models. Researchers noticed significant progress in value-aware accelerators on ASICs, yet we are concerned about the few studies on FPGAs. This paper observed the limitations of implementing bit-level sparsity optimizations using FPGA and proposed a software/architecture co-design solution. Specifically, by introducing LUT-friendly encoding with adaptable granularity and hardware structure supporting multiplication time uncertainty, we achieved a better trade-off between potential redundancy and accuracy with compatibility and scalability. Experiments show that under accurate calculation, PEs are up to 2.2× smaller than bit-parallel ones, and our design boosts performance by 1.04× to 1.74× and 1.40× to 2.79× over bit-parallel and Booth-based designs, respectively.**

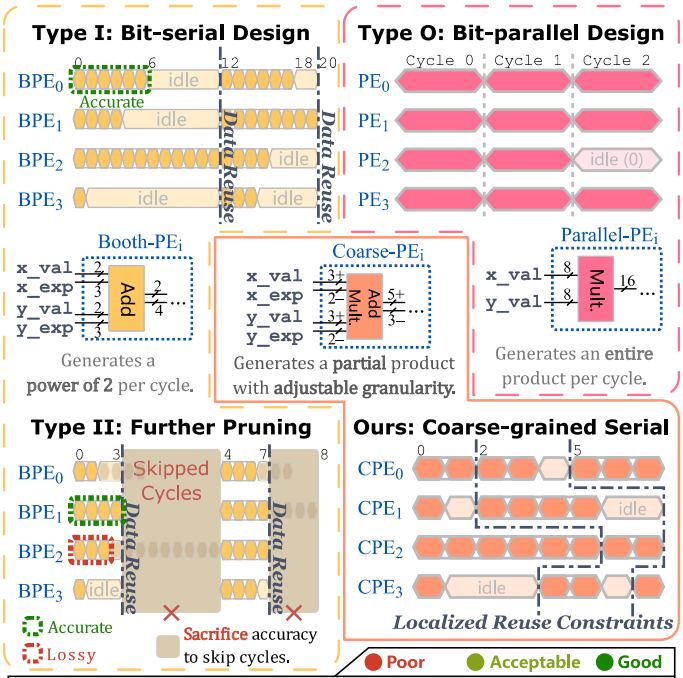
***Index Terms*—Bit-level sparsity, multiply and accumulate, SW/HW co-design, FPGA, look-up table.**

I. Introduction

The flourishing of deep neural networks (DNNs) has been engendering a rapidly escalating hardware demand for computation and storage, posing new challenges to model performance and efficiency during inference. Together with general-purpose CPUs and GPUs, researchers are also vigorously prompting inference acceleration based on ASIC [1]–[6] and FPGA [7]–[10] platforms. Matrix multiplication and convolution dominate over 90% of DNNs [11], with their computations almost entirely composed of multiply-and-accumulate (MAC) operations. The performance of MACs induces crucial impacts on accelerators.

After appropriate training and quantization, the activation or weight data are more likely to be equal to or close to zero [12]. Bit-level sparsity optimization attempts to skip "0" bits within a binary value, Previous works [13]–[17] have given us a convincing demonstration that the distribution characteristics of non-zero values, in addition to strictly zero values, is also plausible for acceleration. The plausibility in calculation turns into reality through bit-serial multipliers based on Booth encoding [2], [4], [7]. Booth encoding eliminates consecutive 0s and 1s, allowing serial multiplication faster, thus surpassing bit-parallel multipliers. Preprocessing methods like Booth encoding, can effectively reveal the bit-sparsity of operands and cope with varying data characteristics among different data sets.

On the one hand, we have perceived that in the former designs, the awareness of bit-sparsity, the certainty of multiplication time, and the accuracy of results can be satisfied at most two out of the three. To preserve the accuracy of serial multipliers, one must tolerate the unpredictable number



Solution(s)	Type O	Type I	Type II	Ours
Bit-Sparsity	None	Max(with optimal booth encoding*)		Consecutive 0/1
Codec Cost	None	High		Low
Accuracy	Loseless	Loseless	Lossy	Loseless
Utilization	High	Low	Relatively High	Relatively High
Bitwidth	Fixed	Variable	Variable	Adaptable

*Optimal Booth Encoding: See Section II-B.

Fig. 1. Comparison of temporal utilization. Multipliers change from bit-parallel (Type O) to bit-serial (Type I/II) after introducing bit-sparsity awareness. The certainty of multiplication time (Type II) and the accuracy of results (Type I) cannot be satisfied simultaneously.

This work is supported in part by the National Key R&D Program of China under Grants 2022YFB4501600 and 2022YFB4501603, in part by the National Natural Science Foundation of China under Grants 61976200 and 62172380, in part by Jiangsu Provincial Natural Science Foundation under Grant BK20241818, in part by Youth Innovation Promotion Association CAS under Grant Y2021121, in part by USTC Research Funds of the Double First-Class Initiative under Grant YD2150002011. *Lei Gong*, *Chao Wang* and Xuehai Zhou are corresponding authors. Email: {leigong0203, cswang, xhzhou}@ustc.edu.cn.

of cycles for non-zero products, leading to the issue of load imbalance [1]–[4]. The derived methods [7], [10], [14], [18], [19] aim to evenly allocate the workload by pruning some of the non-zero bits or trits. However, the pruning refers to secondary quantization, which in turn lowers the performance of models. If it is possible to perform a bit-level customization in compliance with the data characteristics of the target workload, A better trade-off would come to pass for the target model.

On the other hand, innate advantages for customizing bit-sparsity acceleration are in line with the bit-level reconfigurability of FPGAs, anyway there are only a few studies [7], [8], [10]. We are of the opinion that this is caused by the poor resource utilization imputable to the primitive structure of look-up tables (LUTs). Extremely high parallelism in bit transmission, including fan-ins and fan-outs, is typical for bit-serial multiplier logic, which is difficult to deploy efficiently via LUTs. It is adequate to negate the performance of dynamically identifying bit-sparsity at the level of multipliers alone.

Following the observations above, we advanced bit-serial multipliers to a more adaptable co-design solution for encoding and architecture, the three aspects described above. In section III, a compute-oriented, LUT-friendly, and granularity-adjustable encoding is introduced while preserving accuracy. In Section IV's architecture, at the multiplier level, processing elements are coordinated with the encoding. Then, at the cluster level, load-balancing strategies are available for multiplication time uncertainty. As a foundation, we revise approaches and challenges for bit-level sparsity awareness in section II.

II. Background and Motivation

A. Bit-level Sparsity Acceleration

Within DNN inference scenarios, integers and fixed-point numbers are appreciated for their hardware-friendliness, as floating-point numbers' exponential capability is too excessive. The non-uniform distribution of the integers results in less effective information than n bits contained in an n-bit number, exhibiting redundancy within the data.

Per layer precision [20], [21] is a form of bit-level sparsity, since values with smaller bit widths are equivalent to larger ones with bit-sparsity at higher bits. Such quantization method has spawned accelerators that support multiple precisions [22]–[24], which combine their small bit-width multipliers through reconfigurable logic, as shown in Fig. 2(a), belongs to type O in Fig. 1. The layer-wise analysis only reveals the differences in importance between layers, without analyzing the non-uniformity within a layer. To exploit redundancy further, the multipliers should be value-aware at run-time. Specifically, they dynamically adjust the spatial or temporal cost based on the precision of the operands. The spatial redundancy of bit-parallel multipliers, shown in Fig. 2(a), should first be converted into the temporal redundancy of bit-serial multipliers, shown in Fig. 2(b), and only then can it possibly be removed.

However, this approach does not conform to the paradigm of most accelerators. In General, data are arranged regularly, and each value equally possesses the same precision. The schedule for transmission and calculation is predictable across space and time, thus avoiding catastrophic data transmission. Once bit-level sparsity is involved, unpredictable dynamic precisions emerge among different values, leading to unstructured sparsity. A randomly varying temporal cost for each multiplication takes place, and a load imbalance issue occurs under the constraints of data reuse, shown in Fig. 1 type I. It not only negatively impacts inference latency, but exacerbates load imbalance with enhanced reuse and stronger communication constraints, which is detrimental to design scalability. A straightforward way is to directly address the bottlenecks that provoke load imbalance by pruning some lower bits, as shown in Fig. 2(c) and type II in Fig. 1, but it inevitably impairs the performance of the target model. Forcing models to adapt to the accelerator would weaken its compatibility with models into the bargain.

Introducing a value-aware multiplier provides feasibility for revealing bit-sparsity and then acceleration. Unfortunately, such designs have created a conflict between load balancing and lossless computing, hindering the speed and quality of inference. In brief, the awareness of bit-sparsity, the certainty of multiplication time, and the accuracy of the results cannot be achieved at the same time.

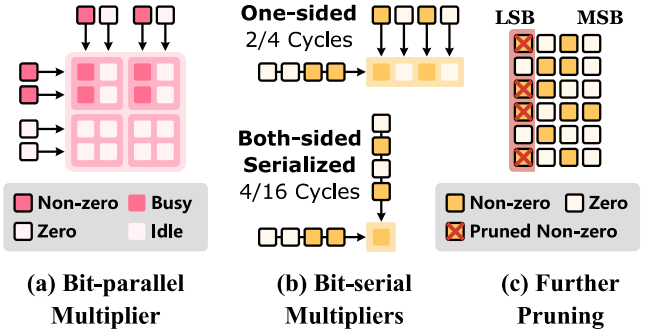


Fig. 2. (a) A reconfigurable parallel multiplier. (b) One-sided and both-sided serial multipliers. (c) Pruning to mitigate multiplication time uncertainty in serial multipliers.

B. Booth Encoding for Preprocessing

A value-aware multiplier design is built on or similar to the serial Booth multiplier. As a means of pre-processing, Booth encoding appears to have sufficient revelation to bit-sparsity. We discuss whether the pre-processing plays a critical role in the conflict described in section II-A.

$$x = \mathrm{val}(B_x) = \sum_{s=0}^{n-1} b_s 2^s \tag{1}$$

Equation (1) shows how the coefficient sequence $B_x = (b_{n-1}, b_{n-2}, ..., b_0)$ maps to the original value x. As shown in Fig. 3(a), the standard integer representation enables only two states $b \in \{0, 1\}$ for each coefficient, while $\{0, -1\}$

for significands. In radix-2 or radix-4 Booth encoding, each coefficient equally enables three states $b \in \{-1, 0, 1\}$. For an n-bit value, since $\left|\{-1,0,1\}^{n+1}\right| \gg |\{0,1\}^n|$, there must be multiple Booth encodings mapping to the same value. It allows us to select an encoding with the smallest non-zero coefficient set, which we call the "optimal Booth encoding".

The optimal Booth encoding performs a precise analysis for the bit-sparsity down to each bit. It exhaustively exposes redundancies arising from non-uniformity. Booth encoding, or any compute-oriented preprocessing method, is usually required to accomplish very quickly. In contrast, in the inference scenario there are lower thresholds for preprocessing, because the encoding overhead is amortized by repeated multiplications.

In opposition, it's expensive to extend the bit width from fixed to variable. Serial multipliers require splitting the coefficient sequence, a process that results in loss of alignment information. We have to append some positional bits, which causes on-chip communication overhead, and nothing to say adding a third state to the coefficients, As for the bit-sparsity analysis down to each bit, it is sometimes an excessive treatment that is of no good or even harmful. For lower bits of larger values, although relatively unimportant, they contain sufficient information, so any preprocessing struggles to analyze redundancy from them. The fluctuation of non-zero term set size is ralated to such excessiveness, as shown in Fig. 3(b), along with underutilization problems under reuse constraints.

C. Bit-serial Multipliers on FPGA

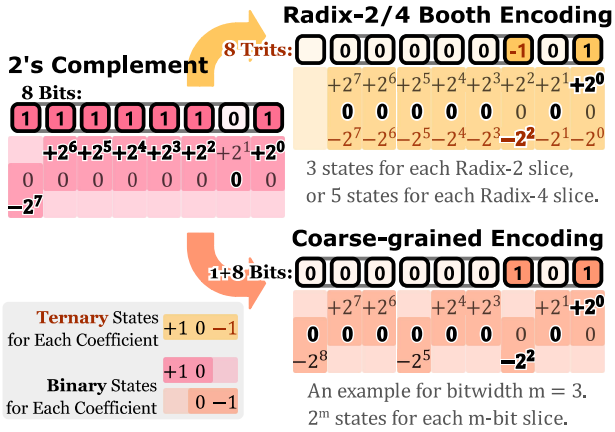
Massive interconnections and intermediate results, as well as simple gates and selectors, are common features of bit-serial multipliers on ASIC platforms. 5 to 6-input LUTs are considered the most suitable choice in common [25], yet when dealing with simple parallel bit operations they are more susceptible to I/O bottlenecks. An intolerable waste arises that a LUT6's functionality is at least the same as a 6-input ROM, but is likely to implement a few basic gates.

The Radix-4 Booth encoding outputs 5 states for each 2 bits instead of 4, inflating the number of bits required for each coefficient to 150%. This leads to a decrease in the information density among on-chip buffers and multiplications. Consequently, we need a preprocessing method that is more friendly to look-up tables. The reconfigurability allows adjusting the granularity of bit-sparsity revelation and switch granularity between target workloads, provides us the opportunity to alleviate the excessive treatment.

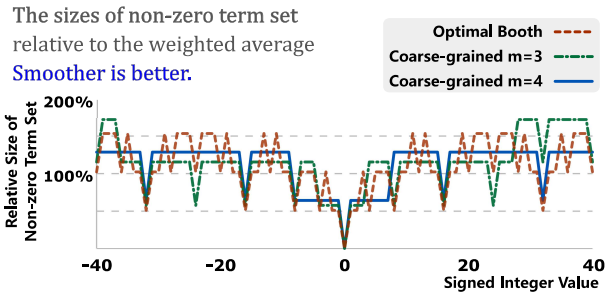
III. Compute-Oriented Codec Method

The preprocessing that reveals bit-sparsity should have the following properties, as specified in section II-A.

- (a) A consistent format for each generated term (or slice).
- (b) The negative correlation between the size of the non-zero term set and the proportion of the corresponding value in a dataset.



(a) Treating Negative Values



(b) Variation of the Size of Non-zero Term Set

Fig. 3. The proposed coarse-grained encoding: (a) introduces fewer additional states, is more LUT-friendly, and (b) generates fewer and smoother numbers of non-zero terms, and simplifies the scheduling.

Furthermore, section II-B and II-C put more potential improvement forward.

- (c) An adaptable granularity for bit-sparsity analysis.
- (d) A higher information density in all terms generated.

An obvious and low-cost approach is to divide the binary representation into slices of length m. For signed integers, it yields one signed term and several unsigned terms, which complicates the preservation of property (a). Alternatively, it can be deduced from property (b) that each term should be signed, otherwise negative values with smaller absolute values cannot be represented by fewer non-zero terms. Simply attaching a duplicated significand to each term would result in redundancy. Hence we consider an approach of embedding rather than attaching sign information.

- ***Serialize.*** Divide the n-bit value into m-bit slices. A term consists of a coefficient $B_{x,i}$ and a shift $s_{x,i} = im$ with its weight $w_{x,i} = 2^{im}$. Perform sign extension on the most significant term if needed.
- ***Embed.*** Flip the state definition of the MSB in each coefficient $B_{x,i}$, i.e., $B_{x,i}[m-1] = b_{im+m-1} \in \{0, -1\}$, without appending to or changing present bits.
- ***Regulate.*** Add $B_{x,i}[m-1]$ to $B_{x,i+1}$ from lower to higher terms. Carry over to the next higher term if an overflow occurs.

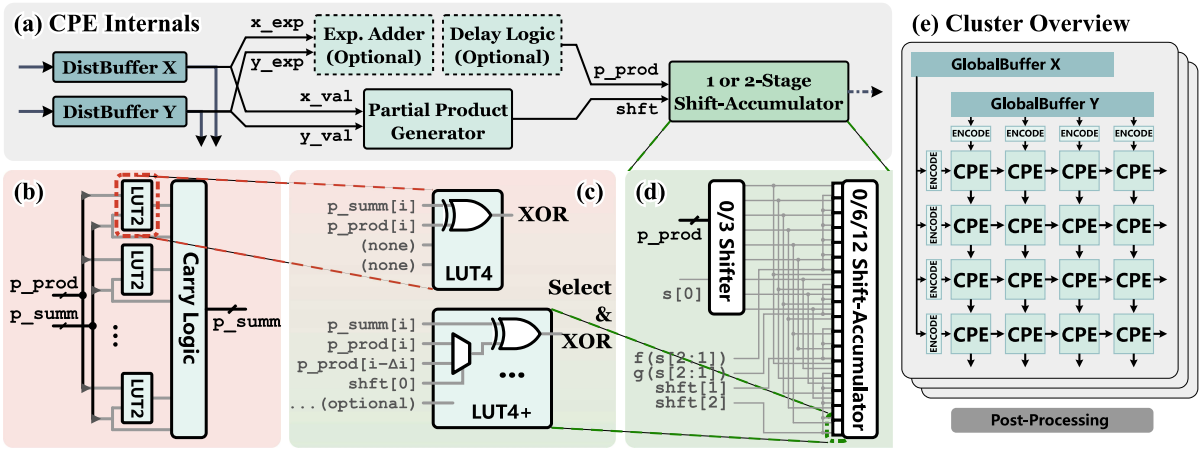


Fig. 4. (a) Internal components of a PE. (b) An adder with fast carry chains. (c) Merging shift logic into LUTs. (d) An example shift-accumulator for Xilinx FPGA. (e) Clusters composed of PEs in (a).

The preprocessing procedure utilizes only the existing bits to represent sign information with no extra bits appending to each term, as shown in Fig. 3(a). Note that the extremes of signed integers (the largest $1/2^{m+1}$ in the range) require an extra bit at the most significant term due to overflow, i.e., at most $\lceil (n+1)/m \rceil$ non-zero terms. All the generated coefficients are represented as signed or 2's complement unsigned integers, with $\sum \mathrm{val}(B_{x,i}) \times w_i = \mathrm{val}(B_x)$. Therefore, we can sum the Kronecker product of the two term sets to complete a multiplication, as shown in (1), thus satisfying property (a).

$$xy = \sum_{i,j} \mathrm{val}(B_{x,i})\mathrm{val}(B_{y,j}) w_{x,i} w_{y,j} \tag{1}$$

The lengths m of partitions of the operands on both sides can vary as integers between 2 and n, modifying the range of coefficients to satisfy property (c). The two bit widths of the operands with a sum of less than 6 is unsuitable for fully utilizing the 6 or more-input LUTs [26]. Accordingly, m should be no less than 3. m_x and m_y for the two operands x and y are preferably be the same or have a larger greatest common divisor, as detailed in section IV-A. For the coarse-grained encoding with bit width $n = 8$ and granularity $m \in \{3, 4\}$, Fig. 3(b) shows the correlation that the size of the non-zero term set varies with the value, compared to the optimal Booth encoding.

Please note that the proposed encoding, while similar to Booth encoding, has essential differences. A coarser-grained encoding, which reduces the mapping space and implies a weaker analysis of redundancy, admittedly loses some revelation in bit-sparsity. Instead, it alleviates the excessive treatment problem in Radix-2 or 4 Booth encoding while maintaining property (b). Moreover, it allows more potential in optimizing scheduling overhead, as Fig. 3(b) also shows that The sizes of the sets no longer fluctuate dramatically. Radix-2^m Booth encoding always defines an additional coefficient state $+2^m$ to avoid carry-overs, and enables the parallelism of multiple-term generations. Even so, one is more concerned with the overall throughput rather than the latency of a single multiplication in intensive and regular operations. It follows that the parallelism of the preprocessing is irrelevant. Such coarse-grained encoding sacrifices latency, in exchange for more compliance with property (d), namely, a more LUT-friendly format for transmission and computation.

IV. Hardware Architecture

A. Processing Unit Design

According to the serialization, PEs are designed to generate partial products first. In each cycle, the distributed buffers inside provide a pair of terms $(B_{x,i}, s_{x,i})$ and $(B_{y,j}, s_{y,j})$. The two coefficients $B_{x,i}$ and $B_{y,j}$ are input to the parallel multiplier, generating a partial product with a bit width of up to m_x+m_y, meanwhile the adder generates the shift $s_{x,i}+s_{y,j}$. Fig. 4(a) shows the execution order for partial products inside a PE. Over multiple cycles, a unit reads and combines from the two buffers inside, and traverses the Descartes product of the non-zero term sets of two operands to make up a complete multiplication. An adder for shifts is not necessary when the number of possible shifts is too small. The partial product and its shift are expected to arrive at the next stage simultaneously, thus delay logic may be required.

In the next stage, the shift of the partial product is variable and unpredictable. LUTs are not good at implementing logic with lots of inputs and outputs, including a parallel-out adjustable shifter and counting buckets. For this reason, we reduce the number of shift operations and the shift is simplified. The number of possible shifts is a decisive factor affecting the resource occupancy of parallel-out shifters. To prevent shifting with more than two levels of LUTs, a larger greatest common divisor for the operands' shifts is suggested. For instance, with two 8-bit operands and their encoding bit width $m_x = m_y = 3$, there are only 5 different possible shifts ($\{0, 3, 6, 9, 12\}$). If $m_x = 3, m_y = 4$, it not only does not decrease but increases to at least 6 ($\{0, 3, 6, 4, 7, 10\}$).

Due to good properties of addition, We only need one accumulator for each inner product. As shown in Fig. 4(b), if

a separate adder is implemented, the relatively simple addition becomes even easier with the help of carry chains. Each LUT is used only as an XOR gate, leaving redundancy in functionality. The spare inputs enable them to perform limited shift operations, as shown in Fig. 4(c). Therefore we merge part of the shifter into the accumulator while keeping another part at a separate level.

Fig. 4(d) the mapping of shift with sign extension to two-level LUT6s on shift granularity $\gcd(m_x, m_y) = 3$ with Xilinx FPGAs. For $\gcd = 4$ and above it comes to be simpler. In contrast, the optimal Booth encoding only guarantees that the shift distance is a multiple of 2 bits.The area of the partial multipliers is positively correlated with the encoding bit width m, while the shift-accumulators are almost unaffected.

B. Array Organization and Communication

As shown in Fig. 4(e), we organize the PEs into output-stationary arrays. In a systolic array, all PEs logically update their states synchronously, and each PE only obtains the most significant information from its neighbors (horizontally, vertically, or diagonally, etc.). After introducing uncertainty in multiplication time, a simple approach is to wait for all PEs in the array to complete their multiplications or inner products. This is absolutely safe but results in poor temporal utilization, as shown in Fig. 5(a) belonging to Fig. 1 type I.

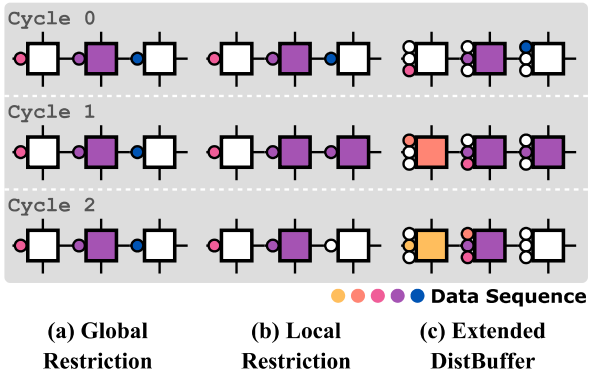


Fig. 5. Assuming that the middle PE encounters a sudden heavy load. (a) Under a global constraint. (b) Downstream PE releases in advance. (c) Upstream PE obtains data in advance.

Here, we adopt some more aggressive but safe communication strategies. Each PE has a set of registers for the operands in both directions X and Y. On this basis, we add a 1-bit connection between the originally connected PEs, allowing the neighbors to be aware of each other's status and decide the data transmission occasions on their own. In other words, the read and write of a buffer are no longer performed simultaneously and the restriction of global synchronization is also lifted, as shown in Fig. 5(b). The sudden occurrence of heavy loads may still hinder transmission, so there is room for further relaxation of communication constraints. One set of registers is feasible be extended to two or more sets, as shown in Fig. 5(c), where local congestion upstream is patched. Furthermore, the extended capacity makes it possible for the time uncertainties of successive multiplications within an inner product to cancel each other out. A larger buffer capacity conduces better temporal utilization, but capped to 100%, hence there is a trade-off between buffer capacity and temporal utilization.

C. Model Compatiability

For the repeatedly reused weights in a model, the encoders for them can be greatly simplified to serializers by performing the *Regulate* step in section III offline in advance. It doesn't contribute to any storage overhead, since the coarse-grained encoding doesn't append any bits to the coefficients, and remains orthogonal to data-level and bit-plane compressions.

The calculation is faster for inputs with fewer non-zero terms. The accelerator supports per-data bit-sparsity, so it naturally supports per-layer precision. Secondary float-point-like quantizations are also available for further speed-up, though the proposed architecture aims to avoid it. The performance under different granularity options can be estimated by calculating the product of the average number of non-zero terms on each layer through software means. Section V-B compares the difference between the estimated and the real.

We do not recommend practices such as combining four 6-bit units into a 12-bit unit, as this would lead to an imbalance load between the four units. Communication strategies in section IV-B are designed for local random fluctuations in bit-sparsity and is powerless against load imbalance per line or column. But in convolution operation, the spatial correlation of images and the im2col algorithm ensure that the data distribution of adjacent windows does not have significant differences.

V. Evaluation Results

A. MAC Units Implementation

In the proposed architecture, small parallel multipliers use the Booth multiplier implementation [27] . For comparison, an accelerator with fixed precision and no additional components for serialization are provided. Our implementation is based on Vivado 2023.1 and Xilinx UltraScale+ series' xczu9eg-ffvb1156-2-e device, running at 400MHz, as shown in Table I. Each PE has four sets of 2-bit control signals, 6-bit address lines, and 6-bit data lines for exchanging data with upstream and downstream. The encoding granularity $m = 2$ is definitely weaker than $m = 3$, so it is not shown. The mapping is also valid for the 6, 7, and UltraScale series with the same LUT6_2 components and similar CLB structure.

In the instantiation of the distributed buffer, for the encoding granularity $m \in 3, 4, 5$, one 32×6-bit simple dual-port RAM provides just enough bit width to cache the terms and length information of the encoding. Since the two operands in an inner product are aligned in the two distributed RAMs, the higher 3-bit read addresses are shared, and the write address reuses the read address upstream. Each unit only needs to generate 7 bits of addresses and 2 bits of control signals per cycle, so as to realize the state machine using only one additional CLB. In such implementation, the options of

TABLE I
IMPLEMENTATION FOR COARSE-GRAINED PES AND ARRAYS

Encoding Granularity	RAM and Controller	Multiplier	Adder	Shifter	Accumulator	Total	Number of Units	Overall LUT Utilization
	(CLBs per Unit)							
$m_x = m_y = 3$	2 (1+1)	0.75*	0.25*	0.625*	3.375*	7	3456	75.6%
$m_x = m_y = 4$	2 (1+1)	1.75*	0.25*	0.625*	3.375*	8	3072	75.9%
$m_x = m_y = 5$	1.875 (1+0.875)	4.5*	0	0	3.375*	10	2496	74.9%
None (8-bit)	N/A					18	1456	73.8%

*Shares SLICEs.

$m_x = 3, m_y = 6$ or $m_x = 4, m_y = 8$ has almost no practical value in reducing control overhead.

For the target models, due to the similarity between bit-level and data-level sparsity, the offline load balancing [28] is applied. The differential convolution [2] is also applied if the boost is positive. We choose ResNet-18 and VDSR quantized to 8-bit for comparison with a relatively evident difference in data distribution. Per-layer precision is banned for generality and conciseness. Fig. 6(a) shows the MAC per second difference of the bit-parallel design in each convolutional layer. VDSR, with stronger bit-sparsity, performs significantly better than ResNet-18, and their advantageous granularity m are different. VDSR achieves an average of 1.74× at $m = 4$, while the lowest is 1.04×, with some layers performing even worse than the bit-parallel design. This is reasonable because without any information redundancy, the theoretical speedup is 0.4×, conversely it reaches up to 2.2× at applying per layer precision. The unsatisfactory result with $m = 3$ indicates that the redundant analysis of the mantissa-like LSBs is usually excessive.

Our implementation doesn't make any special treatment for zeros. In other words, multiplications involved with zeros are the same as non-zero ones and take one cycle. For highly sparse models, a data-level sparsity optimization is better. The unstructured bit-level sparsity is relatively mild and is hardly worth any compression representation. Besides, optimization for bit-serial multipliers may be diluted by the logic introduced at the data level, i.e., too sparse to optimize.

B. Communication Strategies Comparison

If a PE does not perform multiplication in a cycle, or performs one results in "0", we call it idle, or macroscopically temporal underutilization. The proportion of idle time is equivalent to the ratio of actual cycles to the estimated cycles based on the product of the average size of the non-zero term set.

Fig. 6(b) shows the variation in temporal utilization rate under the joint influence of encoding granularity and the size of distributed buffers. For ResNet-18, compared to the naive strategy of a low-power design [4], the improvement in the utilization rate brought by relaxing communication constraints ranges from 1.40× to 2.79×. It only needs two extra CLBs, since the distributed buffers are easier to fully exploit the functionality of LUTs. A LUTRAM with a depth of 32 and capable of holding eight integers is more than enough. The potential improvement is no more than 16%, while adding

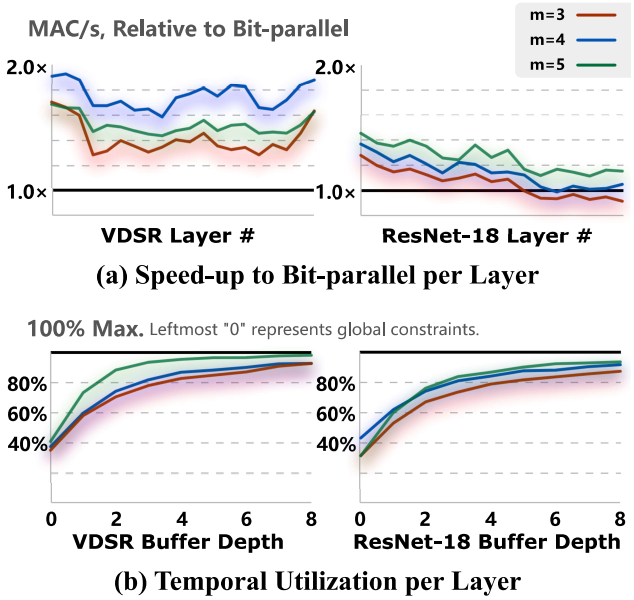


Fig. 6. (a) The advantageous encoding granularity m is larger. This indicates that MSBs mainly cause redundancy. (b) Buffer depth 8 is more than enough.

a SLICE would increase the area cost by 5% to 7%, thus one LUTRAM is already at the sweet point. It is feasible to estimate the performance directly by getting some samples and calculating the product of average encoding lengths, since the error is very likely within 10%. Buffers composed of 4-input LUTs with depth 16 are also sufficiently powerful, making the duty ratio close to 80%.

VI. CONCLUSIONS

Our software/hardware co-design achieves a better trade-off, that is, sacrificing some bit-sparsity revelation in exchange for the alleviation of excessive treatment, the balance of workloads, and the accuracy of calculations. The LUT-friendliness is embodied in two main aspects: one is the higher information density of encoding for better LUT-based transmission and calculation, and the other is the sufficient functionality of LUTRAMs to address multiplication time uncertainty. Enhancements in communication strategies achieved a temporal utilization increase of up to 2.79× compared to the naive strategy, at a tolerable cost of 2 CLBs per unit. The proposed FPGA-based implementation achieves up to 1.74× area efficiency under general-purposed integer quantization workloads, demonstrating compatibility with the design paradigm.

REFERENCES

[1] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 382–394.

[2] M. Mahmoud, K. Siu, and A. Moshovos, "Diffy: a déjà vu-free differential deep neural network accelerator," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 134–147.

[3] L. Cavigelli, G. Rutishauser, and L. Benini, "Ebpc: Extended bit-plane compression for deep neural network inference and training accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 723–734, 2019.

[4] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 304–317.

[5] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 688–698.

[6] F. Liu, N. Yang, H. Li, Z. Wang, Z. Song, S. Pei, and L. Jiang, "Spark: Scalable and precision-aware acceleration of neural networks via efficient encoding," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 1029–1042.

[7] H. Kung, B. McDanel, and S. Qian Zhang, "Term revealing: Furthering quantization at run time on quantized dnns," *arXiv e-prints*, pp. arXiv–2007, 2020.

[8] X. Dai, Y. Chen, and M. S. Abdelfattah, "Kratos: An fpga benchmark for unrolled dnns with fine-grained sparsity and mixed precision," in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, 2024, pp. 156–163.

[9] K. Sun, M. Koch, Z. Wang, S. Jovanovic, H. Rabah, and S. Simon, "An fpga-based residual recurrent neural network for real-time video super-resolution," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 4, pp. 1739–1750, 2022.

[10] S.-E. Chang, Y. Li, M. Sun, R. Shi, H. K.-H. So, X. Qian, Y. Wang, and X. Lin, "Mix and match: A novel fpga-centric deep neural network quantization framework," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 208–220.

[11] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[12] (2021) Achieving fp32 accuracy for int8 inference using quantization aware training with nvidia tensorrt. [Online]. Available: https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/

[13] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[14] F. Liu, W. Zhao, Z. Wang, Y. Chen, Z. He, N. Jing, X. Liang, and L. Jiang, "Ebsp: evolving bit sparsity patterns for hardware-friendly inference of quantized deep neural networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 259–264. [Online]. Available: https://doi.org/10.1145/3489517.3530660

[15] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-plane compression: Transforming data for better compression in many-core architectures," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 329–340.

[16] L. Cavigelli, G. Rutishauser, and L. Benini, "Ebpc: Extended bit-plane compression for deep neural network inference and training accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 723–734, 2019.

[17] Y.-C. Lo and R.-S. Liu, "Bit-serial cache: Exploiting input bit vector repetition to accelerate bit-serial inference," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.

[18] S. Q. Zhang, B. McDanel, H. T. Kung, and X. Dong, "Training for multi-resolution inference using reusable quantization terms," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 845–860. [Online]. Available: https://doi.org/10.1145/3445814.3446741

[19] X. Zhao, Y. Wang, C. Liu, C. Shi, K. Tu, and L. Zhang, "Bitpruner: Network pruning for bit-serial accelerators," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

[20] C. Sakr and N. Shanbhag, "An analytical method to determine minimum per-layer precision of deep neural networks," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 1090–1094.

[21] C. Tang, K. Ouyang, Z. Wang, Y. Zhu, W. Ji, Y. Wang, and W. Zhu, "Mixed-precision neural network quantization via learned layer-wise importance," in *Computer Vision – ECCV 2022*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds. Cham: Springer Nature Switzerland, 2022, pp. 259–275.

[22] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 764–775.

[23] S. Ryu, H. Kim, W. Yi, and J.-J. Kim, "Bitblade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[24] S. Lee and Y. Kim, "Booth fusion: Efficient bit fusion multiplier with booth encoding," in *2020 International SoC Design Conference (ISOCC)*, 2020, pp. 73–74.

[25] (2006) Fpga architecture white paper. [Online]. Available: https://cdrdv2-public.intel.com/771003/wp-01003.pdf/

[26] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes, and B. Popa, "Arithmetic core generation using bit heaps," in *2013 23rd International Conference on Field programmable Logic and Applications*, 2013.

[27] M. Kumm, S. Abbas, and P. Zipf, "An efficient softcore multiplier architecture for xilinx fpgas," in *2015 IEEE 22nd Symposium on Computer Arithmetic*, 2015, pp. 18–25.

[28] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, pp. 151–165. [Online]. Available: https://doi.org/10.1145/3352460.3358291

Uncertainty-Aware Energy Management for Wearable IoT Devices with Conformal Prediction

Dina Hussein*, Chibuike E. Ugwu*, Ganapati Bhat, and Janardhan Rao Doppa
School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, 99164

***Abstract*—Wearable internet of things (IoT) are transforming diverse healthcare applications including rehabilitation, vital symptom monitoring, and activity recognition. However, the small form-factor of wearable devices constrains the battery capacity and the operating lifetime, thus requiring frequent recharging or battery replacements. Frequent recharging and battery replacement leads to lower quality of service and user satisfaction. Harvesting energy from ambient sources to augment the battery has emerged as an effective solution to improving its operating lifetime. However, ambient energy sources are highly stochastic, making energy management challenging. Prior approaches typically use point predictions for estimating future energy and do not explicitly account for the uncertainty. In strong contrast to prior approaches, this paper presents a conformal prediction-based method for future energy harvest that provides small uncertainty regions with provable coverage guarantees (true output is within the uncertainty region). The uncertainty regions over energy harvest are then leveraged in an energy management algorithm that employs Monte Carlo sampling to evaluate the quality of multiple decisions with varying energy harvests. The decisions are then combined using a lightweight machine learning model to make an energy management decision that is close to the optimal. Experiments on two diverse real-world datasets with about 10 users show that conformal prediction achieves more than 90% coverage with tight prediction intervals; and the energy management algorithm produces decisions that are on average within 2 J of an optimal Oracle, thus showing its effectiveness in improving the quality of service.**

***Index Terms*—Energy Management, Conformal Prediction, IoT Devices**

I. Introduction

Internet of Things (IoT) devices have the potential to enable transformative applications including digital agriculture, health monitoring, wide area monitoring, and smart homes [1–6]. Energy harvesting and management are promising technologies to improve the self-sustainability of IoT devices [7–9]. However, ambient energy sources are highly stochastic with wide variations across locations, time, seasons, and user behavior [7]. Similarly, energy demand from the applications can vary across time due to sensing requirements, user activities, and environmental conditions. For instance, sensors need higher sampling rates and power when a user is performing exercise when compared to sleeping. Accounting for these variations at runtime is crucial to ensure sustainability of the device.

Achieving self-sustainability in IoT devices consists of two key steps: 1) future energy availability prediction and 2) management of the device energy. Future energy availability predictions are important to understand the variability of energy resource in the system. Accurate predictions ensure that the system is able to maintain sufficient energy reserves and does not go into low power mode. The energy management (EM) algorithms must consider the uncertainty in predicted future energy dynamics to make optimal decisions to achieve self-sustainability and maximize application quality [10, 11]. To this end, this paper develops novel approaches for energy predictions with *correct-by-design uncertainty bounds* and *uncertainty-aware* EM decisions.

Energy harvest (EH) predictor must satisfy two main conditions: 1) it must provide high prediction accuracy with respect to ground-truth; and 2) it must produce uncertainty bounds for each prediction with guaranteed high coverage (true value lies within the interval) and small prediction intervals (tight uncertainty bounds). To satisfy these conditions by design, we propose a multi-target conformal prediction (MTCP) approach to obtain small uncertainty regions with coverage guarantees. The key idea behind MTCP is to extend the idea of single target conformal prediction [12] to multiple target variables (one for each future EH interval) and construct a joint uncertainty region. MTCP is a simple and low-overhead approach that allows us to easily incorporate domain constraints (e.g., energy is not negative). By considering multiple target variables, the MTCP approach is able to provide high-quality prediction intervals that adapt to temporal variations in EH data.

Prediction of future EH is not sufficient for sustainable operation of IoT devices. This is because the predictions and uncertainty bounds must be used within the EM algorithm to obtain energy allocation or budget for each decision interval. The energy allocation must account for variations in ambient energy and device energy requirements. To this end, we formulate a constrained optimization problem to obtain energy allocation bounds that satisfy application requirements, maximize application quality, and maintain energy sustainability while accounting for uncertainty in EH.

The EM algorithm employs Monte Carlo sampling under overhead constraints to evaluate multiple EM decisions as per varying EH predictions and associated uncertainty. The sampling is done such that it covers the entire uncertainty region while not exceeding the overhead budget for EM. The EM decisions must be combined to obtain a single energy budget that is as close to an optimal decision made by an Oracle that has access to ground-truth EH values. To this end, we use a lightweight machine learning (ML) model that takes candidate decisions from the Monto Carlo sampling and provides a decision that aims to mimic the Oracle for each interval. In summary, the proposed approach first predicts uncertainty bounds for EH using MTCP and then performs Monte Carlo sampling using EH uncertainty bounds to obtain EM decisions.

We validate the proposed approach using EH and activity data from two diverse real-world datasets [13, 14]. Activity information from the two datasets is combined with solar irradiation data for outdoor and indoor conditions to obtain ground truth EH data. The data are then used to validate the proposed MTCP for EH uncertainty quantification and the uncertainty-aware EM approach. The evaluation shows that the proposed approach produces decisions that are on average within 2 J of an optimal Oracle. End-to-end experiments on a prototype IoT device show that the approach consumes at most 88 mJ energy in the worst-case scenario, with lower consumption under typical conditions while improving the performance by at least 25% compared to baselines.

In summary, this paper makes the following contributions:

- A novel multi-target conformal prediction (MTCP) algorithm that provides guaranteed coverage and tight uncertainty bounds

*Dina Hussein and Chibuike E. Ugwu contributed equally to this work.

for future EH over multiple time horizons. To the best of our knowledge, this is the first approach that has applied MTCP in the context of EH prediction.

- Novel EM algorithm that leverages Monte Carlo sampling to account for uncertainty in the EH to make decisions that are on average within 2 J of an optimal Oracle. The Monte Carlo sampling allows us to smartly trade-off the overhead of EM while accounting for uncertainty in the EH.
- Comprehensive evaluations using two real-world activity datasets with indoor and outdoor EH values. The evaluations show that the proposed approach achieves better performance that at least 25% higher than baseline.

II. Related Work

Accurate prediction of future EH is vital for optimal energy allocation. Previous studies have applied analytical and ML methods to address this task [9, 15–18], primarily focusing on generating single-point predictions of future EH. While these approaches provide estimates, they lack uncertainty quantification, which is essential for reliable EM decisions in wearable IoT systems.

Recent research has explored ensemble methods to construct reliable prediction intervals by combining the mean (μ) and variance (σ) estimators to form prediction intervals [19, 20]. However, these intervals are often overly conservative and frequently fail to guarantee the necessary coverage which is critical for real-world EM applications. Thus, there remains a need for methods that provide tight uncertainty regions while providing coverage guarantees.

We propose to utilize conformal prediction (CP) to overcome the limitations of prior EH prediction approaches, particularly for uncertainty quantification [21]. CP is a distribution-free uncertainty quantification framework for constructing prediction intervals with guaranteed finite-sample coverage for any given predictive model [12, 22–31]. CP constructs prediction intervals by defining a non-conformity score between predictions and ground truth, computing scores on calibration data, and determining a quantile threshold based on a user-defined error rate (say 5%). Motivated by the need for uncertainty regions in EH predictions, this work explores the under-studied application of CP for multi-target regression.

EM for wearable IoT devices is a well-studied problem [7, 8, 32–37]. Prior approaches for EM include linear programming, model predictive control, and reinforcement learning. These approaches aim to perform EM such that the device performance is maximized while ensuring long-term operation. However, prior approaches fail to consider the uncertainty in future EH when making decisions. Consequently, they are unable to react to sudden changes in EH behavior (e.g., cloudy days followed by sunny days). This leads to sub-optimal performance for the device. The proposed work aims to precisely overcome the limitations of prior approaches by reasoning about uncertainty in multiple future intervals, thus making it more robust when compared to baseline approaches.

III. Background and Problem Setup

We consider a wearable IoT system that consists of multiple sensors and harvesters, as shown in Figure 1. We assume that the device integrates a rechargeable battery. We also assume that wearable devices in this work integrate EH from body motion and ambient light to augment the battery.

The goal of EH is to direct energy to the battery for future usage thereby minimizing the frequency of recharging operations. We can potentially eliminate recharging needs if the energy consumed in a given horizon, such as a day, is equal to the harvested energy.

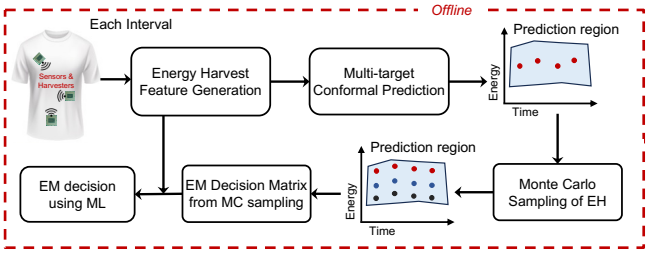


Fig. 1: Overview of our uncertainty-aware energy management approach.

This will lead to energy-neutral operation (ENO) and significantly reduce recharging needs. However, achieving ENO is challenging in practice due to the following reasons.

Uncertainty in EH: Ambient sources of energy are highly stochastic in nature due to variations in lighting conditions, user activity levels, and location. We must quantify the uncertainty in the EH to avoid over- or under-allocation of energy.

Uncertainty-Aware EM: EM algorithms must carefully determine the energy budget in each time horizon to balance two objectives: 1) the maximization of application accuracy and quality of service, and 2) minimizing energy consumption and maintaining sufficient battery levels at all time horizons to successfully maintain ENO. Achieving the two objectives is challenging because the EM algorithm must reason about uncertainty in future EH so that battery energy reserves are effectively managed and not exhausted.

The proposed approaches in this paper precisely aim to handle the above challenges by formulating and solving the following problems for uncertainty quantification and EM.

A. Uncertainty Quantification for EH

We consider that the EM decisions are made with a fixed time horizon of one day with each day divided into T equal length intervals [38]. Our goal is to design a function f that takes a set of input features X and provides *multiple* EH predictions for intervals t to $t+H$, where H is the number of future time intervals. The function f must also provide accurate uncertainty bounds that meet user-specified coverage guarantees at runtime.

B. Energy Management Policy

The goal of the EM policy π is to take EH predictions from f as input and provide an energy budget for interval t that maximizes application quality and energy-sustainability. The policy π must make decisions that also account for the uncertainty in the EH by intelligently sampling the uncertainty. To this end, our goal is to design a policy π that intelligently samples the uncertainty regions in EH while maintaining ENO and maximizing application quality.

IV. Uncertainty Quantification for Energy Harvest Prediction

CP [12, 22–25] is a robust framework that quantifies uncertainty and provides rigorous coverage guarantees for any predictive black-box model. Given a predictive model f, n calibration examples with p features and d targets $\{X_i, Y_i\}_{i=1}^{n}$: $X_i \in \mathbb{R}^p$ and $Y_i \in \mathbb{R}^d$ (which were not used during training), a user-specified error rate α (say 5%), a scoring function $S(X_i, Y_i)$ that measures the discrepancy between the predicted value $f(X_i)$ and the ground truth Y_i (e.g., absolute residual), and a new test input X_{n+1}, instead of a conventional single point prediction, CP generates a prediction set $\mathcal{R}(X_{n+1}) \subseteq \mathbb{R}^d$ that confidently ensures that the true output Y_{n+1} falls within it with a probability of at least $1-\alpha$ on average.

CP can be used to construct prediction sets for classification tasks or prediction intervals for regression tasks. In this work, we focus on the regression setting with multiple target variables, as our goal is to quantify the prediction uncertainty for EH across a prediction horizon $H > 1$ time periods. For a univariate target ($Y_i \in \mathbb{R}^H$) with $H = 1$, the uncertainty region $\mathcal{R}(X) \subseteq \mathbb{R}$ is represented as an interval defined by its lower and upper bounds. However, for multi-dimensional targets ($H > 1$), the uncertainty region $\mathcal{R}(X) \subseteq \mathbb{R}^H$ becomes an H-dimensional region, capturing the joint uncertainty.

A. Multi-Target Conformal Prediction (MTCP)

We propose a lightweight MTCP approach designed to construct uncertainty regions for EH with guaranteed theoretical and empirical coverage with low overhead suited for resource-constrained devices.

Given a user-specified error rate, α, and a prediction horizon H, MTCP employs Conformalized Quantile Regression (CQR) [12] to construct base prediction intervals for each target horizon $h \in \{1, \ldots, H\}$. The adjusted error rate for each target is defined as $\beta = \alpha/H$. The prediction interval for each target is constructed as:

$$\mathcal{R}_h(X) = [\hat{q}^h_{\beta/2}(X), \hat{q}^h_{1-\beta/2}(X)],$$

where $\hat{q}^h_{\beta/2}(X)$ and $\hat{q}^h_{1-\beta/2}(X)$ represent the lower and upper quantile estimates for the h-th target variable, respectively.

To reduce the uncertainty regions, we incorporate the known non-negativity constraint of EH (energy is not negative) into MTCP. Specifically, the prediction interval for each target variable is adjusted to fall in the positive region:

$$\mathcal{R}^+_h(X) = \big[\max\{0, \hat{q}^h_{\beta/2}(X)\}, \hat{q}^h_{1-\beta/2}(X)\big],$$

This constraint reduces the size of the uncertainty regions without compromising the coverage guarantees. The final constrained H-dimensional uncertainty region becomes:

$$\mathcal{R}^+(X) = \bigcap_{h=1}^{H} \mathcal{R}^+_h(X)$$

For a new test input X_{n+1}, the coverage inequality in Equation 1 is guaranteed to hold marginally:

$$\mathbb{P}[Y_{n+1} \in \mathcal{R}^+(X_{n+1})] \geq 1 - \beta H \tag{1}$$

where $\mathcal{R}^+(X) \subseteq \mathbb{R}^H$ and $\beta H = \alpha$. By constructing uncertainty regions in this manner, the MTCP approach ensures that the true EH values lie within the predicted uncertainty region with a high probability of at least $(1 - \alpha)$. This method provides tighter uncertainty regions and remains computationally lightweight, making it well-suited real-time EM.

V. Uncertainty-Aware Energy Management

A. Problem Formulation

We employ a common formulation used to achieve ENO in energy-harvesting devices, first introduced by Kansal et al. [7]. The optimization problem can be formulated as follows:

$$\max. \quad Q(\mathbf{E_A}) = \sum_{t=0}^{T-1} \beta^t \ln\left(\frac{E^t_A}{M_E}\right) \quad \text{s. t.} \tag{2}$$

$$E^{t+1}_B = E^t_B + \eta \mathcal{E}^t_H - E^t_A - E_o, \ 0 \leq t \leq T-1 \tag{3}$$

$$E^{t+1}_B \geq E_{min} \quad 0 \leq t \leq T-1 \tag{4}$$

$$E^T_B \geq E_{target} \tag{5}$$

Objective: The key objective is to maximize the quality of service Q represented by the sum of logarithms of the energy allocation (budget) E^t_A in each interval t. Intuitively, higher energy allocations lead to higher quality of service since devices can have higher sampling rates and run more complex algorithms. The objective is scaled by a constant M_E to ensure that the objective Q is positive only when the device is able to perform a minimum level of sampling and processing for the application.

Constraints: The problem has three main constraints. First, the constraint in Equation 3 specifies that the battery level at the beginning of an interval $t + 1$ is governed by the battery in the previous interval t, stochastic EH $\mathcal{E}^t_H$ with efficiency η, energy allocation E^t_A, and overhead of EM decision-making E_o. The second constraint ensures that battery levels in the device are always greater than a minimum value of E_{min}. The E_{min} reserve ensures additional energy for the device when there are mispredictions in EH or incorrect EM decisions. Finally, the constraint in Equation 5 specifies that the battery at the end of the horizon, i.e. E^{t+1}_B is greater than a target value of E_{target}. Consequently, if we ensure that the battery energy at the beginning and end of a horizon is greater than E_{target}, the device will achieve ENO.

Solution Approach: The EM problem is a convex optimization problem with a concave objective and linear constraints [39]. The optimization problem can be solved using convex optimization solvers using interior-point methods [39] or with iterative solvers. Iterative solvers are attractive since their execution can be fine-tuned to balance accuracy and overhead. Inspired by this, we propose to utilize the iterative gradient projection (IGP) algorithm proposed in [40] to solve the optimization problem.

Even with an iterative algorithm, it is challenging to solve the EM problem optimally at runtime due to the unavailability of future EH values. At the same time, we can obtain the upper bound on the quality of EM decisions by solving the problem *offline* with actual EH values. Indeed, we utilize the optimal solutions with actual EH values as the **Oracle** policy to compare the effectiveness of the proposed and baseline EM policies.

B. Monte Carlo Sampling of EH Uncertainty Regions

The uncertainty in EH values must be handled carefully to make decisions that maximize the QoS while maintaining ENO. Algorithm 1 shows the proposed approach to handle the uncertainty in future EH. At a given interval t, the algorithm takes EH predictions and uncertainty regions from MTCP, overhead budget for EM, and current battery status as inputs. The uncertainty bounds provide a region of possible values for EH in the next H intervals. Intuitively, the region is a H dimensional hyperrectangle as explained in the previous section. We must sample this multi-dimensional space to evaluate various EM decision choices. To this end, the primary loop in the algorithm randomly samples an H-dimensional vector from the EH uncertainty region. It is critical to sample the prediction region uniformly to ensure that the EM decisions are not skewed by low or high EH predictions. To this end, we utilize Monte Carlo sampling to obtain random EH prediction samples while maintaining a wide coverage across the uncertainty regions.

The H-dimensional vector is then used as the EH prediction for intervals t to $t+H$ before utilizing IGP to obtain the EM decisions for intervals t to $T-1$. That is, we solve the EM problem from interval t until the end of the current horizon. The decisions are stored in a matrix for further analysis before making the final EM decision. The process is repeated until the energy budget for making EM decisions is exhausted. At the end of iterations, the matrix with

Algorithm 1: Monte Carlo Sampling and EM Decisions

Input: MTCP EH uncertainty region for intervals t to $t+H$, Overhead budget E_o, Features X and Battery E_B^t
$M_{E_A} \leftarrow$ Initialize decision matrix
while $E_o \geq 0$ **do**
  $[\hat{E}_H^t, \ldots, \hat{E}_H^{t+H}] \leftarrow$ Randomly sample EH
  $[\tilde{E}_A^t, \ldots, \tilde{E}_T^{T-1}] \leftarrow$ Solve EM problem with EH samples
  $M_{E_A} \leftarrow [M_{E_A}; [\tilde{E}_A^t, \ldots, \tilde{E}_T^{T-1}]]$
  $E_o \leftarrow E_o - E_{IGP}$
end
Use M_{E_A} and X as input for ML model to obtain EM decision
return EM decision

decisions from each EH sampled vector is provided as output of the algorithm for final decision-making.

C. Lightweight ML Model for Oracle Tracking

The wearable IoT device must make a single decision for the energy budget based on the decisions provided by Monte Carlo sampling. A straightforward approach is to randomly choose a decision among the decisions provided by the sampling. However, the random decision does not consider the optimality of the decisions and closeness to the optimal Oracle. To this end, we propose to leverage the trajectory of decisions provided by the Monte Carlo sampling and MTCP features to train an ML model that aims to obtain decisions that are close to the Oracle. Key components and insights behind the ML model are described in the following.

Input Features: The trajectory evaluations performed by the Monte Carlo sampling include potential decisions for intervals t to $T-1$. Each row of the evaluations considers different levels of uncertainty and we can use the average of these decisions as a feature for the ML model. In particular, the average of decisions allows the feature space to be of the same size when the variable number of samples are obtained in the previous section. The features used for predicting EH values additionally provide insight into past EH values, therefore they can aid in making the final EM decision. Consequently, the feature set for EM decision-making consists of the EH prediction features X and $\hat{E}_A^{t+1} - \hat{E}_A^{t+H}$, where $\hat{E}_A^{t+1}$ represents the average of potential EM decisions.

Training Target: Our goal is to obtain decisions that are close to the Oracle without full system knowledge of the future. Therefore, we use the Oracle EM decision as the training target for the ML model. Specifically, we obtain the Oracle solution offline and use it as the target for the ML model.

Model Structure: Any supervised learning algorithm can be utilized to obtain the EM decisions. In this work, we employ fully connected neural networks due to their simplicity and strong performance in diverse scenarios. Additional details on the neural network structure and its configuration are provided in Section VI-A5.

VI. Experimental Results

A. Experimental Setup

1) Wearable IoT Device Model: Our system design focuses on a wearable IoT device that is equipped with five energy harvesters and multiple sensors. It includes piezoelectric sensors strategically placed at four locations on the knees and elbows to efficiently harvest motion energy. Additionally, the device integrates an SP3-37 [41] flexible photovoltaic (PV) cell for light and solar EH.

2) Datasets: The approach aims to deliver uncertainty-aware predictions of energy harvest in both indoor and outdoor environments. Therefore, we leverage activity data from two publicly available datasets: 1) ARAS [13] and 2) Mannheim [14]. The ARAS dataset includes activity traces collected over 30 days from four users residing in two houses, while the Mannheim dataset provides data from six users over a two-week period.

The ARAS and Mannheim datasets provide user activity data spanning one month and two weeks, respectively. While useful, these durations are insufficient for our goal of evaluating accuracy across different seasons and times of the year. To address this limitation, we extend both datasets to cover a six-year period (2015–2020). This is achieved by shuffling and augmenting the original activity data, allowing us to simulate long-term usage patterns and conduct comprehensive evaluations. The activity data are combined with typical EH values from motion and ambient light irradiance to obtain actual EH values in each interval [9, 42].

3) Data Splitting: The six-year energy dataset is divided into three subsets: 1) training, 2) validation, and 3) testing. The data from 2015 is allocated for training, 2016 is reserved for validation, and remaining data are used for testing.

4) Proposed CP Parameters: The parameters for MTCP consist of the base CQR prediction model architecture, prediction horizon H, and error rate $\alpha = 0.1$ corresponding to a 0.9 expected marginal coverage level. The model architecture is three layers and 64 neurons per layer network. Given prediction horizon $H \in \{2, 3, 4, 5\}$, for each target $h \in \{1, ..., H\}$, the error rate is set to $\beta = 0.1/H$.

5) Proposed EM Parameters: The proposed EM framework parameters consist of the decision intervals, battery constraints, and ML model structure. The EM horizon spans 24 hours, divided into 24 decision intervals throughout the day. The battery energy target is set to 100 J, with a minimum energy threshold of 10 J. The framework incorporates a discount factor γ of 0.99. The utility ME is defined as eight. The ML model used in the EM framework consists of four fully connected layers with 32 neurons per layer.

6) Evaluation Metrics: The MTCP framework is evaluated for the coverage and size of the uncertainty region. Next, we evaluate the quality of the individual EM decisions by comparing it to the optimal Oracle that uses actual EH values. We use the mean absolute error (MAE) compared to the Oracle to evaluate the proposed EM approach and baseline algorithms. Finally, the cumulative QoS Q over the horizon is used to evaluate application performance.

B. Baseline Methods for Comparison

1) Baseline Point Prediction Method: The most common approach to perform EM is to use point predictions of EH to make decisions using iterative algorithms. While any point prediction approach is applicable, we utilize the average of last three days of actual EH values as the point prediction in this approach due to its simplicity and ease of computation.

2) Baseline uncertainty region with Ensemble Method: The ensemble method uses a collection of random forests to first obtain EH predictions for each interval. The individual predictions are then combined to form prediction intervals by using the mean (μ) and variance (σ) of predictions. Specifically, we can obtain the bounds as: $(\mu \pm \sigma)$. The final uncertainty region for all horizons is obtained as the H-dimensional intersection of all prediction intervals.

3) Random Choice Energy Decision (MTCP-Random): One of the straightforward methods is to choose the final EM decision randomly from the range of decisions provided by the Monte Carlo sampling. To this end, the MTCP-Random serves as a baseline

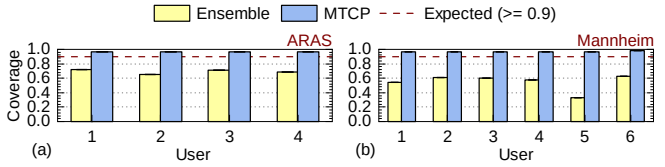


Fig. 2: Coverage comparison between Ensemble and MTCP prediction uncertainty regions for ARAS and Mannheim datasets across all users in Horizon 4. Similar trends are observed across all other prediction horizons.

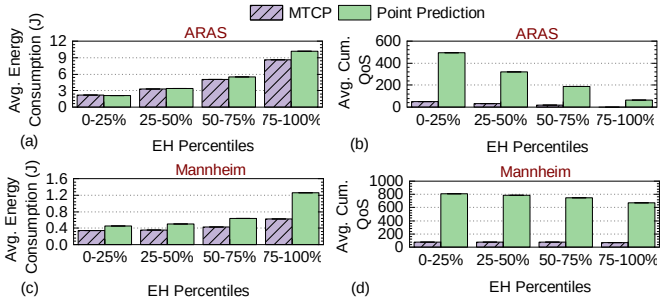


Fig. 3: Comparison of average energy consumption and absolute of average cumulative utility of MTCP, and point prediction with user one in ARAS and Mannheim datasets. Lower cumulative values lead to better quality since the logarithms are typically negative.

TABLE I: Mean of MAE for MTCP, MTCP-Random, Ensemble-ML, Ensemble-Random, and Point Prediction methods with different prediction horizons across years for user 1 of ARAS dataset. The algorithms are named with the convention of {EH-Pred}-{EM}.

Method	Pred. Horizon H	Year 2017	2018	2019	2020
MTCP	**H = 2**	**1.96**	**1.99**	**1.87**	**1.80**
	H = 3	**1.96**	**1.99**	**1.93**	**1.82**
	H = 4	**1.98**	**2.00**	**1.91**	**1.93**
	H = 5	**1.98**	**2.03**	**1.91**	**1.87**
MTCP-Random	H = 2	2.58	2.65	2.49	2.37
	H = 3	2.60	2.65	2.58	2.45
	H = 4	2.77	2.74	2.69	2.61
	H = 5	2.72	2.75	2.66	2.64
Ensemble-ML	H = 2	2.66	2.72	2.64	2.63
	H = 3	2.90	3.04	2.95	2.86
	H = 4	3.09	3.24	3.08	3.01
	H = 5	3.17	3.25	3.12	3.14
Ensemble-Random	H = 2	5.44	5.60	5.36	5.32
	H = 3	6.19	6.23	6.00	6.22
	H = 4	6.33	6.65	6.22	6.42
	H = 5	6.47	6.74	6.45	6.57
Point Prediction	H = 2	2.60	2.62	2.49	2.35
	H = 3	2.60	2.62	2.49	2.35
	H = 4	2.60	2.62	2.49	2.35
	H = 5	2.60	2.62	2.49	2.35

by randomly choosing an energy allocation in the range provided by the sampling. We use this baseline to highlight the benefits of incorporating ML-based decision-making in the proposed method.

C. Evaluation of the Proposed MTCP Approach

A critical aspect of predicting future EH is the ability to manage the prediction error effectively and to quantify the uncertainty associated with predictions. Achieving a coverage level that meets or exceeds the user-specified expectation is essential for reliable and accurate future energy allocation. To this end, this section evaluates the coverage with different prediction horizons H. Figure 2 shows the coverage for the proposed MTCP and ensemble methods. Figure 2 shows that the ensemble method fails to achieve the desired coverage level of 0.9, while the proposed MTCP approach exceeds the desired coverage for all users in both datasets, thus showing the effectiveness of our approach.

D. Evaluation of the Proposed EM Approach

1) Energy Management with Point Predictions: We compare the performance of the proposed approach with point prediction. Figure 3 compares the average energy allocation and the absolute value of the average cumulative utility for both approaches using data from user one in the ARAS and Mannheim datasets. Each bar represents a range of EH, allowing for an analysis of trends across low to high-energy availability scenarios. The results highlight key energy efficiency and QoS differences between the proposed approach and the point prediction approach. Specifically, the proposed approach demonstrates improved quality consistency across percentiles compared to point prediction. Specifically, the point prediction approach has worst utility even when consuming more energy due to its inability to account for uncertainty, leading to over-allocation and battery depletion. Overall, these results demonstrate the importance of considering uncertainty EM.

2) Validation with Different Prediction Horizons: Next, we evaluate the proposed approach with different prediction horizons given as input to the CP algorithm. Specifically, we vary the prediction horizon H from two to five and evaluate the EM performance. Table I presents a comparison of the MAE for EM decisions for all the EM algorithms from 2017-2020 for user 1 of the ARAS dataset. We note that other users show similar performance.

The table shows that the proposed MTCP approach demonstrates consistent performance with low MAE values across all horizons and years. For instance, the MAE for $H = 2$ ranges between 1.87 and 1.96 J across the four years, showing minimal variation. This highlights the robustness of our method when adjusting the prediction horizon. In contrast, baseline methods exhibit higher MAE values overall, with the Ensemble-Random approach showing the largest errors, ranging from 5.32 to 6.74 across horizons and years. The results indicate that our MTCP method consistently outperforms baseline approaches with different prediction horizons.

3) Evaluation of EM Decisions over a Day: This section analyzes the accuracy of the EM approaches against the optimal Oracle for a given user and day. Specifically, we obtain the energy decisions and battery energy levels over a day from all approaches and compare them to the optimal Oracle. Figure 4 shows energy allocations and battery energy levels in (a) and (b) for user four in ARAS dataset. It also shows energy allocations and battery energy levels in (c) and (d) for user six in Mannheim dataset.

As expected, MTCP maintains levels closer to Oracle, demonstrating its ability to make effective energy decisions. In contrast, baseline methods show larger deviations, with MTCP-Random and Point Prediction exhibiting the most significant mismatches. Overall, the figure illustrates that MTCP provides superior energy allocations and battery management compared to baseline approaches.

4) Comparison of Error with Different EM Approaches: EH patterns fluctuate over the year due to seasonal variations. Therefore, we demonstrate the effectiveness of our approach regardless of the season and the year. To this end, Figure 5 shows the average MAE and variance in each month across years. We can see that the MAE for ARAS dataset varies for different approaches with Ensemble-Random approach showing the highest error. In contrast, MTCP has the lowest MAE among all approaches. Similarly, MTCP has the lowest error for Mannheim dataset as well.

Next, we analyze the MAE across years. Specifically, Figures 6

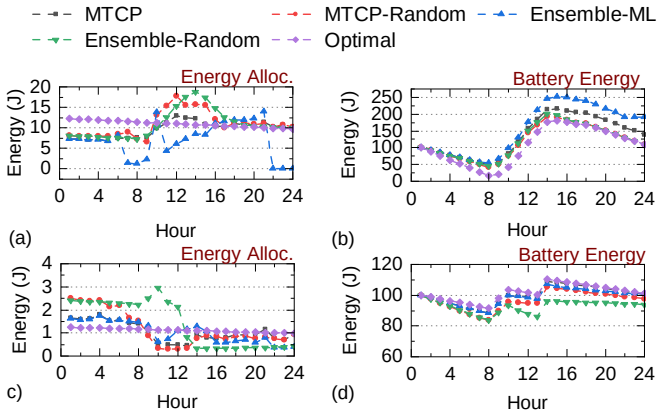


Fig. 4: Comparison of energy allocation and battery energy with MTCP, MTCP-Random, Ensemble-ML, Point Prediction, and Optimal for (a) and (b) user four in ARAS and (c) and (d) user six in Mannheim dataset for the day June 28th, 2017.

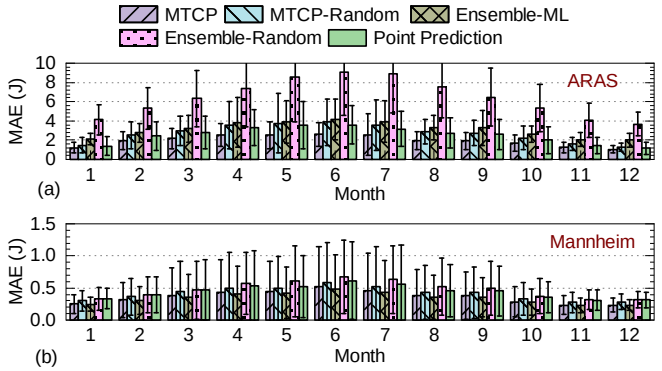


Fig. 5: Monthly Mean Absolute Error (Mean and standard deviation) of MTCP, MTCP-Random, Ensemble-ML, Ensemble-Random, and Point Prediction with user one in (a) ARAS (b) Mannheim dataset.

and 7 show the yearly MAE across different users in both datasets. The figures show the mean of MAE along with its standard deviation. MTCP consistently outperforms baseline approaches across all years, thus reflecting its ability to adapt effectively to changing EH patterns. These results underscore the robustness of MTCP in providing accurate predictions across diverse users and years.

E. Implementation Overhead

We employ the TI-CC2652R microcontroller [43] to measure the energy consumption and execution time of the proposed algorithms: MTCP, the IGP algorithm, and the ML model. The MTCP and ML models are implemented as multi-layer perceptrons. The MTCP algorithm includes constructing uncertainty regions based on neural network outputs. Measurements show that MTCP requires approximately 58 ms for execution, with an associated energy cost of 0.6 mJ per operation. The IGP algorithm, which iteratively optimizes energy usage, incurs a higher computational load due to its gradient-based approach. Execution of a single IGP operation requires approximately 300 ms, with an energy consumption of 8.6 mJ. Finally, the overhead measurement of the ML model shows that a single inference takes 188 ms, consuming 2.0 mJ of energy.

These overheads are modest in comparison to the operational timeframes and energy budgets of typical IoT applications. The overhead is included as part of the optimization in Equation 3 for all approaches. Moreover, the overhead can be minimized at runtime

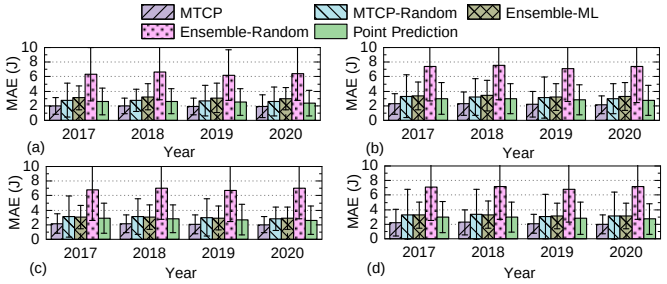


Fig. 6: Yearly Mean Absolute Error (Mean and standard deviation) of MTCP, MTCP-Random, Ensemble-ML, Ensemble-Random, and Point Prediction with different users in ARAS dataset.

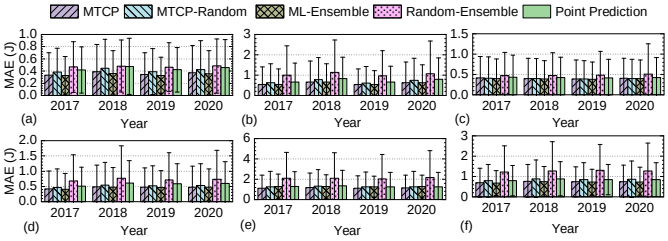


Fig. 7: Yearly Mean Absolute Error (Mean and standard deviation) of MTCP, MTCP-Random, Ensemble-ML, Ensemble-Random, and Point Prediction with different users in Mannheim dataset.

optimizing the Monte Carlo sampling and the IGP algorithm. In summary, the overhead of these algorithms remains within acceptable limits for resource-constrained IoT devices, ensuring their practical applicability in real-world scenarios.

VII. Conclusion

Wearable and IoT devices hold great potential for applications such as health monitoring and digital agriculture. Despite this promise, their adoption is often limited by the constraints of their small battery capacities. Ambient EH provides a viable solution to extend the battery life of these devices. However, the unpredictable nature of EH poses significant challenges for EM in wearable devices. Efficient EM relies on accurate predictions of future EH and the understanding of uncertainty associated with those predictions. Therefore, this paper proposed a novel energy management framework that leverages conformal prediction (CP) to provide uncertainty-aware EH forecasts over multiple prediction horizons. To enhance decision-making, an ML model is employed to select the optimal EM decision from multiple CP-generated uncertainty regions, balancing energy efficiency and device performance. The proposed approach outperforms baseline methods by ensuring reliable EM decisions, even under highly stochastic EH scenarios.

Acknowledgment

This work was supported in part by NSF CAREER award CNS-2238257 and in part by the AgAID AI Institute for Agriculture Decision Support, supported by the United States Department of Agriculture (USDA) - National Institute of Food and Agriculture (NIFA) award #2021-67021-35344.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] A. J. Espay *et al.*, "Technology in Parkinson's Disease: Challenges and Opportunities," *Movt. Disorders*, vol. 31, no. 9, pp. 1272–1282, 2016.

[3] J.-F. Daneault, "Could Wearable and Mobile Technology Improve the Management of Essential Tremor?" *Frontiers in neurology*, vol. 9, pp. 257:1–257:8, 2018.

[4] D. Hussein, T. Belkhouja, G. Bhat, and J. R. Doppa, "Reliable Machine Learning for Wearable Activity Monitoring: Novel Algorithms and Theoretical Guarantees," in *Proc. Int. Conf. on Comput.-Aided Design*, 2022, pp. 33:1–33:9.

[5] D. Hussein, T. Belkhouja, G. Bhat, and J. Doppa, "Sensor-Aware Data Imputation for Time-Series Machine Learning on Low-Power Wearable Devices," *ACM Trans. Design Autom. Electr. Syst.*, vol. 30, no. 1, pp. 1–27, 2025.

[6] D. Hussein, T. Belkhouja, G. Bhat, and J. R. Doppa, "Energy-Efficient Missing Data Recovery in Wearable Devices: A Novel Search-Based Approach," in *Int. Symp. on Low Power Electron. and Des. (ISLPED)*, 2023, pp. 1–6.

[7] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power Management in Energy Harvesting Sensor Networks," *ACM Trans. Embedd. Comput. Syst.*, vol. 6, no. 4, p. 32, 2007.

[8] M. Odema, N. Rashid, and M. A. Al Faruque, "Energy-Aware Design Methodology for Myocardial Infarction Detection on Low-Power Wearable Devices," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2021, pp. 621–626.

[9] Y. Tuncel, S. Bandyopadhyay, S. V. Kulshrestha, A. Mendez, and U. Y. Ogras, "Towards Wearable Piezoelectric Energy harvesting: Modeling and Experimental Validation," in *Int. Symp. on Low Power Electron. and Des. (ISLPED)*, 2020, pp. 55–60.

[10] D. Hussein, N. Yamin, and G. Bhat, "EMI: Energy Management Meets Imputation in Wearable IoT Devices," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 11, pp. 3792–3803, 2024.

[11] N. Yamin and G. Bhat, "Indoor–Outdoor Energy Management for Wearable IoT Devices With Conformal Prediction and Rollout," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 11, pp. 3370–3381, 2024.

[12] Y. Romano, E. Patterson, and E. Candes, "Conformalized Quantile Regression," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[13] H. Alemdar, H. Ertan, O. D. Incel, and C. Ersoy, "ARAS Human Activity Datasets in Multiple Homes with Multiple Residents," in *Int. Conf. Perv. Comput. Tech. Health. and Work.*, 2013, pp. 232–235.

[14] T. Sztyler, J. Carmona, J. Völker, and H. Stuckenschmidt, "Self-tracking Reloaded: Applying Process Mining to Personalized Health Care from Labeled Sensor Data," in *Trans. Petri Nets and other Models Concur.* Springer, 2016, pp. 160–180.

[15] N. Yamin and G. Bhat, "Uncertainty-Aware Energy Harvest Prediction and Management for IoT Devices," *ACM Trans. Design Autom. Electr. Syst.*, vol. 28, no. 5, pp. 1–33, 2023.

[16] A. Cammarano, C. Petrioli, and D. Spenza, "Pro-Energy: A Novel Energy Prediction Model for Solar and Wind Energy-Harvesting Wireless Sensor Networks," in *Int. Conf. on Mobile Ad-Hoc and Sensor Syst.*, 2012, pp. 75–83.

[17] J. R. Piorno, C. Bergonzini, D. Atienza, and T. S. Rosing, "Prediction and Management in Energy Harvested Wireless Sensor Nodes," in *Int. Conf. Wireless Comm., Vehicular Tech., Info. Theory and Aerospace & Electron. Syst. Tech.*, 2009, pp. 6–10.

[18] Z. Wan, Y. Tan, and C. Yuen, "Review on Energy Harvesting and Energy Management for Sustainable Wireless Sensor Networks," in *2011 IEEE 13th Int. Conf. on Communication Technology*, 2011, pp. 362–367.

[19] M.-H. Roy and D. Larocque, "Prediction Intervals with Random Forests," *Statistical Methods in Medical Research*, vol. 29, no. 1, pp. 205–229, 2020.

[20] D. L. Shrestha and D. P. Solomatine, "Machine Learning Approaches for Estimation of Prediction Interval for the Model Output," *Neural Networks*, vol. 19, no. 2, pp. 225–235, 2006.

[21] G. Narang, R. Ayoub, M. Kishinevsky, J. R. Doppa, and P. P. Pande, "Uncertainty-Aware Online Learning for Dynamic Power Management in Large Manycore Systems," in *Int. Symp. on Low Power Electron. and Des. (ISLPED)*, 2023, pp. 1–6.

[22] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic Learning in a Random World*. Springer, 2005, vol. 29.

[23] Y. Romano, M. Sesia, and E. Candes, "Classification with Valid and Adaptive Coverage," *Advances in Neural Inf. Processing Sys.*, vol. 33, pp. 3581–3591, 2020.

[24] I. Gibbs, J. J. Cherian, and E. J. Candès, "Conformal Prediction with Conditional Guarantees," *arXiv preprint arXiv:2305.12616*, 2023.

[25] R. J. Tibshirani, R. Foygel Barber, E. Candes, and A. Ramdas, "Conformal Prediction under Covariate Shift," *Advances in Neural Information Proc. Syst. (NeurIPS)*, vol. 32, 2019.

[26] Q. Regression, *Handbook of Quantile Regression*. CRC Press: Boca Raton, FL, USA, 2017.

[27] R. Koenker and G. Bassett Jr, "Regression Quantiles," *Econometrica: Journal of the Econometric Society*, pp. 33–50, 1978.

[28] S. Ghosh, T. Belkhouja, Y. Yan, and J. R. Doppa, "Improving Uncertainty Quantification of Deep Classifiers via Neighborhood Conformal Prediction: Novel Algorithm and Theoretical Analysis," in *Proc. of AAAI Conf.*, 2023, pp. 7722–7730.

[29] S. Ghosh, Y. Shi, T. Belkhouja, Y. Yan, J. Doppa, and B. Jones, "Probabilistically Robust Conformal Prediction," in *UAI Conf.*, ser. Proc. of Machine Learning Research, vol. 216. PMLR, 2023, pp. 681–690.

[30] Y. Shi, S. Ghosh, T. Belkhouja, J. Doppa, and Y. Yan, "Conformal Prediction for Class-wise Coverage via Augmented Label Rank Calibration," in *Advances in Neural Information Processing Sys. (NeurIPS)*, 2024.

[31] H. Shahrokhi, D. R. Roy, Y. Yan, V. Arnaoudova, and J. R. Doppa, "Conformal Prediction Sets for Deep Generative Models via Reduction to Conformal Regression," 2025. [Online]. Available: https://arxiv.org/abs/2503.10512

[32] C. M. Vigorito, D. Ganesan, and A. G. Barto, "Adaptive Control of Duty Cycling in Energy-Harvesting Wireless Sensor Networks," in *Proc. Conf. on Sensor, Mesh and Ad Hoc Comm. and Networks*, 2007, pp. 21–30.

[33] T. Basaklar, Y. Tuncel, and U. Y. Ogras, "tinyMAN: Lightweight Energy Manager using Reinforcement Learning for Energy Harvesting Wearable IoT Devices," in *tinyML Research Symp.*, 2022, pp. 1–7.

[34] M. Geisler *et al.*, "Human-Motion Energy Harvester for Autonomous Body Area Sensors," *Smart Materials and Structures*, vol. 557, no. 1, p. 012024, 2017.

[35] G. Bhat, J. Park, and U. Y. Ogras, "Near-Optimal Energy Allocation for Self-Powered Wearable Systems," in *Proc. Int. Conf. on Comput.-Aided Design*, 2017, pp. 368–375.

[36] N. Yamin, G. Bhat, and J. R. Doppa, "DIET: A Dynamic Energy Management Approach for Wearable Health Monitoring Devices," in *2022 Des., Autom. & Test in Europe Conf. & Exhibition (DATE)*, 2022, pp. 1365–1370.

[37] D. Hussein, G. Bhat, and J. R. Doppa, "Adaptive Energy Management for Self-Sustainable Wearables in Mobile Health," in *Proc. AAAI Conf.*, vol. 36, no. 11, 2022, pp. 11 935–11 944.

[38] T. Huynh, M. Fritz, and B. Schiele, "Discovery of Activity Patterns using Topic Models," in *Proc. of the 10th Int. Conf. on Ubiquitous Computing*, 2008, pp. 10–19.

[39] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[40] N. Karakoç *et al.*, "Multi-Layer Decomposition of Network Utility Maximization Problems," *IEEE/ACM Trans. on Netw.*, vol. 28, no. 5, pp. 2077–2091, 2020.

[41] FlexSolarCells, "SP3-37 Datasheet," 2013, https://bit.ly/3dcJ0lK, accessed 3/28/2021.

[42] A. Andreas and T. Stoffel, "NREL Solar Radiation Research Laboratory (SRRL): Baseline Measurement System (BMS); Golden, Colorado (Data); NREL Report No. DA-5500-56488," 1981, accessed 3/20/2023.

[43] Texas Instruments Inc., "CC2652R Microcontroller," [Online] https://www.ti.com/product/CC2652R, accessed August 12, 2022, 2018.

DM-Tune: Quantizing Diffusion Models with Mixture-of-Gaussian Guided Noise Tuning

Pouya Haghi*, Ali Falahati*, Zahra Azad*, Chunshu Wu*, Ruibing Song*, Chuan Liu*, Ang Li†‡, Tong Geng*

*University of Rochester, Rochester, NY
†Pacific Northwest National Laboratory, Richland, WA
‡University of Washington, Seattle, WA
*{phaghi, afalahat, zazad, cwu88, rsong10, cliu81, tgeng}@ur.rochester.edu, †ang.li@pnnl.gov

***Abstract*—Diffusion models have become essential generative tools for tasks such as image generation, video creation, and inpainting, but their high computational and memory demands pose challenges for efficient deployment. Contrary to the traditional belief that full-precision computation ensures optimal image quality, we demonstrate that a fine-grained mixed-precision strategy can surpass full-precision models in terms of image quality, diversity, and text-to-image alignment. However, directly implementing such strategies can lead to increased complexity and reduced runtime performance due to the overheads of managing multiple precision formats and casting operations. To address this, we introduce *DM-Tune*, which replaces complex mixed-precision quantization with a unified low-precision format, supplemented by noise-tuning, to improve both image generation quality and runtime efficiency. The proposed noise-tuning mechanism is a type of fine-tuning that reconstructs the mixed-precision output by learning adjustable noise through a parameterized nonlinear function consisting of Gaussian and linear components. Key steps in our framework include identifying sensitive layers for quantization, modeling quantization noise, and optimizing runtime with custom low-precision GPU kernels that support efficient noise-tuning. Experimental results across various diffusion models and datasets demonstrate that DM-Tune not only significantly improves runtime but also enhances diversity, quality, and text-to-image alignment compared to FP32, FP8, and state-of-the-art mixed-precision methods. Our approach is broadly applicable and lays a solid foundation for simplifying complex mixed-precision strategies at minimal cost.**

***Index Terms*—Diffusion Models, Quantization, Mixed-precision, GPU.**

I. Introduction

Diffusion models serve as potent tools for tasks like image generation, video creation, and inpainting, yet they demand significant computational and memory resources [1]–[5]. For instance, generating a single image using Stable Diffusion XL (SDXL) [6], which has approximately 10 billion parameters, requires over two minutes on an A100 GPU, highlighting the urgency for efficiency enhancements.

Recent AI hardware advancements [7], especially GPUs, often rely on low-precision computation to improve performance. For instance, Nvidia's upcoming Blackwell GPUs support a variety of floating-point (FP) formats (from FP16, and BF16, down to FP8, and FP4), yielding a vast search space for efficient execution of deep learning models.

Quantization leverages low-precision hardware to reduce memory and computation costs, but fully quantizing diffusion models to low precision (e.g., 8-bit) often results in low-quality images. Previous research explored mixed-precision strategies, such as intra-layer and timestep-aware methods [8]–[10]. However, these approaches face two main challenges.

1) Full-precision fallacy: It is traditionally believed that full-precision computation offers the highest accuracy. Our research shows that an FP-based mixed-precision (FP-MP) quantization strategy can outperform full-precision in image generation quality, providing more detail and better prompt alignment. Fig. 1 (a) compares images from FP32 and FP-MP (BF16 and FP8) using a Stable Diffusion model. Diffusion models absorb quantization noise into their inherent noise, potentially improving metrics. Unlike previous integer-based methods, our FP approach enhances performance. We compare with PTQ4DM [9], finding that integer-based strategies produce lower quality images. *2) Mixed-precision overhead:* However, mixed-precision can be slower than full-precision due to complex strategies and casting overheads, with low-precision resources underutilized.

To solve these problems, we propose *DM-Tune*, a framework that first finds an efficient FP-MP strategy that outperforms full-precision by progressively introducing **controlled noise**. It then replaces the FP-MP model with a **unified low-precision** model and a **noise-tuning** head to optimize quality and speed. Noise-tuning fine-tunes the model by adding a guided noise head, with minimal overhead, as only the head's parameters are trained and applied once before inference.

To achieve this goal, three challenges must be addressed. First, the vast search space for the optimal FP-MP strategy. Second, effectively expressing the noise-tuning operator. Third, noise-tuning slows the model inference due to memory demands. To resolve the first challenge, we identify sensitive layers and then propose two novel techniques to enhance the quality: *prompt-aware* and *timestep-aware quantization*. Second, we recognize that multiple overlapping Gaussians are needed as nonlinear functions to recover FP-MP output. Finally, we provide a highly optimized GPU kernel that fuses matrix multiplication in low-precision with nonlinear Gaussian terms. This approach reduces runtime and enhances image quality over state-of-the-art (SOTA) and full-precision models (Fig. 1 (b) and (c)). Our main contributions are as follows:

- We show that a novel mixed-precision strategy can

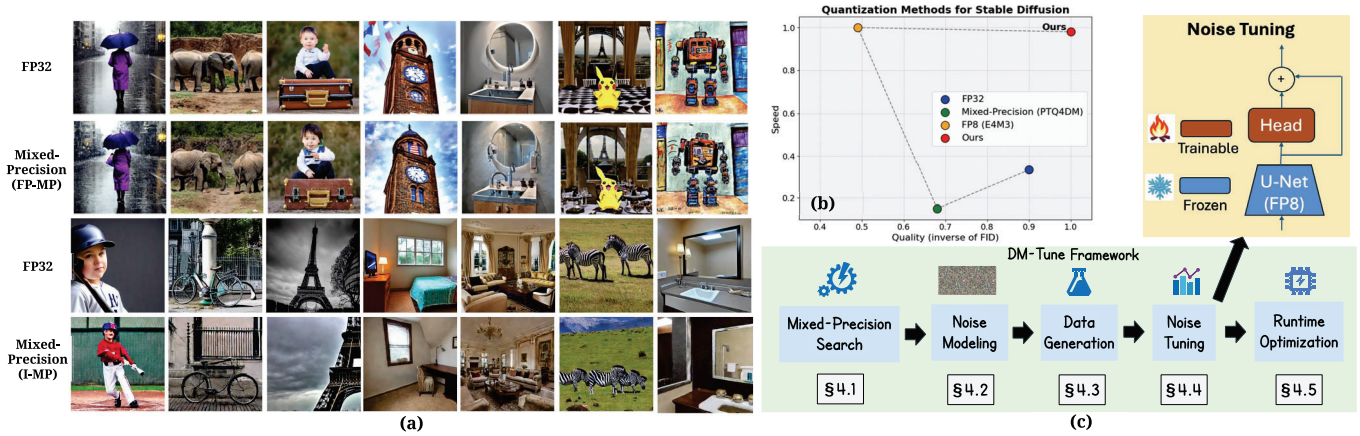


Fig. 1. (a) Generated images in FP32, FP-based mixed-precision (mixed BF16 and FP8), and integer-based mixed precision (Q-Diffusion) using a fixed seed. FP-MP offers greater detail and better prompt alignment compared to FP32. For example, in the prompt "*a view of a multi-tiered clock tower with a US flag on the top*," FP32 misplaces the flags, leaving them suspended in the sky, whereas FP-MP correctly positions a flag at the top of the tower. In contrast, I-MP struggles with image regeneration, particularly when generating human faces. (b) Comparing image quality and speed of our approach with PTQ4DM, FP32, and FP8. (c) Overview of DM-Tune.

outperform full-precision image generation quality and diversity for different types of diffusion models.

- We introduce noise-tuning, a data-free technique for running models in low-precision at high speed with mixed-precision quality.
- We develop highly optimized GPU kernels that fuse low-precision matrix multiplication with nonlinear functions.
- Experimental results show that our approach improves the runtime of prior art by 5.2× while improving image generation quality and diversity.

II. Background

Diffusion Models: Diffusion models generate images using a Markov chain process. Initially, a forward diffusion process adds Gaussian noise to the data $x_0 \sim q(\mathbf{x})$ over T steps, resulting in progressively noisier samples $\mathbf{x}_1, \ldots, \mathbf{x}_T$:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{1}$$

Here, $\beta_t \in (0,1)$ is a variance schedule that determines the intensity of Gaussian noise at each step. As $T \to \infty$, $\mathbf{x}_T$ converges to an isotropic Gaussian distribution.

The backward process removes noise from a sample drawn from the Gaussian noise input $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ to generate high-fidelity images. Since the actual reverse conditional distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is unknown, diffusion models sample from a learned conditional distribution:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_\theta(\mathbf{x}_t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{2}$$

$$\tilde{\mu}_\theta(\mathbf{x}_t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) \tag{3}$$

where $\alpha_t = 1-\beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. The variance $\Sigma_\theta(\mathbf{x}_t, t)$ can either be reparameterized or set to a constant schedule σ_t. When using a constant schedule, $\mathbf{x}_{t-1}$ is given by:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z} \tag{4}$$

where $\epsilon_\theta(\mathbf{x}_t, t)$ is the noise estimation model output at timestep t. The U-Net architecture [11] is predominantly used in designing the noise estimation model. For further details, we refer readers to [12]. This work focuses on quantization of the U-Net during the inference.

Evaluation of Diffusion Models: Evaluating diffusion models differs from other deep learning models due to potential biases and the inadequacy of single metrics like accuracy. Issues such as memorization and mode collapse affect evaluation [13]. We address these by using the *DINOv2-ViT* model as a feature extractor, rather than *Inception-V3*, and employ a variety of metrics. We measure performance with Fréchet Inception Distance (FID), Kernel Distance (KD), and Sliced-Wasserstein Distance (SW) [14]. We also evaluate prompt alignment with CLIP, and assess image quality, diversity, and authenticity using precision, density, recall, coverage, and authenticity metrics [15]. Lower scores are better for FID, KD, and SW, while higher scores are better for the other metrics.

III. Motivation

We summarize the key motivations of this work.

1) Mixed-precision outperforms full-precision: Contrary to traditional belief, FP-MP can surpass full-precision in diffusion models for two reasons. First, the stochastic nature of diffusion models means some seeds yield better results. Second, quantization noise merges with scheduler noise, beneficially altering the diffusion path. We can demonstrate this by rewriting (4) for the quantized model:

$$\begin{aligned}\mathbf{x}'_{t-1} &= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}'_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\left(\epsilon_\theta(\mathbf{x}'_t, t) + \Delta\epsilon_\theta(\mathbf{x}'_t, t)\right)\right) \\ &\quad + \sigma_t\mathbf{z} \\ &= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}'_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}'_t, t)\right) \\ &\quad - \frac{\beta_t}{\sqrt{\alpha_t(1-\bar{\alpha}_t)}}\Delta\epsilon_\theta(\mathbf{x}'_t, t) + \sigma_t\mathbf{z}\end{aligned} \tag{5}$$

where $\mathbf{x}'_t$ and $\Delta\epsilon_\theta(\mathbf{x}'_t, t)$ are the quantized input and U-Net output quantization error at timestep t, respectively. We find mixed-precision benefits unique to floating-point formats (FP-MP), as integer-based mixed precision (I-MP) shows little improvement in image generation due to limited dynamic range and clipping. Thus, maintaining quality with integer quantization requires a complex mixed-precision strategy.

2) Mixed-precision slows down: Mixed-precision designs (I-MP and FP-MP) can be slower than full-precision due to

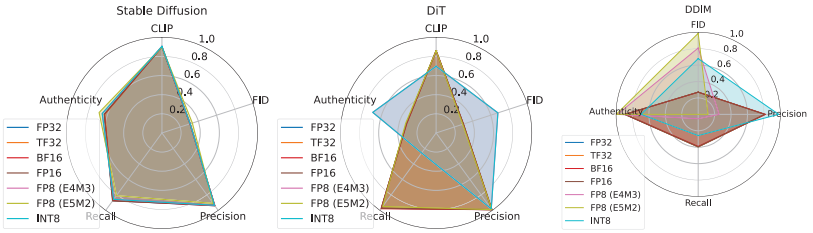


Fig. 2. Evaluating three diffusion models for different data formats.

complex quantization strategies. Therefore, GPUs often rely on unified low precision computation to enhance performance. We found that using unified low-precision for the entire U-Net while adding a nonlinear function with a negligible number of parameters is enough to recover FP-MP output. Our proposed FP-MP will be introduced at the end of Sec. IV-A.

IV. DM-Tune

In this section, we describe the components of DM-Tune framework. Our methodology provides a generalized and scalable solution that can be adapted to different models, datasets, and quantization techniques.

A. Mixed-Precision Search

In this step, we first select two precisions out of all supported data formats in GPUs. Subsequently, we provide our FP-MP quantization methodology to outperform full-precision.

1) Precision Selection: GPU vendors now support multiple data formats like FP32, FP16, BF16, FP8 (E4M3), and FP8 (E5M2). We simplify precision selection to two levels: *low* and *high*. Through experiments on three diffusion models (see Fig. 2), we find 16-bit quantization offers a significant speedup with comparable performance to 32-bit. BF16 generally outperforms FP16, making it our choice for *high-precision*. For *low-precision*, FP8 (E4M3) is preferred over FP8 (E5M2) for better performance since reducing exponent bits too much degrades image quality, while reducing mantissa bits enhances diversity. However, a 2-bit mantissa in 8-bit formats is insufficient.

2) Search Space Reduction: The challenge of mixed-precision search arises from the vast search space: with two formats (BF16, FP8), T timesteps, and L layers, it results in $2^{(T\times L)}$ possibilities. To address this, we reduce the space by classifying layers as either sensitive or insensitive. Sensitive layers require high precision to maintain model quality, while insensitive layers can be quantized to low precision without significant performance loss.

3) Sensitivity Criteria: To identify sensitive layers, we use two criteria: *(1) Range*: Layers with large ranges risk errors from value clipping. *(2) Standard Deviation (STD)*: High STD layers face significant quantization errors due to low precision. We sort layers by these criteria to guide quantization, as detailed in Algorithm 1.

4) Selecting Cutoff Point: After sorting layers by sensitivity, we determine the cutoff between *high* and *low* precision using a *binary search*. Evaluating each point directly is too time-consuming, so we start at the midpoint of the sensitivity list, checking model performance. If quality is below the threshold, we check the midpoint of the upper half; otherwise, the lower half. This continues until narrowed to one layer. Fig. 4 shows cutoff points for different models: DiT needs few high-precision layers, while DDIM requires most. Although this approach helps us achieve near-full-precision quality, it alone may not be sufficient to surpass it.

Algorithm 1 DM-Tune Algorithm for Identifying Sensitive Layers

Input: Calibration dataset, MAX_FLT (maximum allowable value). **Output:** `sens_list`
`sens_list` ← [], Shuffle the calibration dataset.
for each batch of samples in the calibration dataset **do**
  Randomly select a seed.
  Calculate the running average of overflow ratios for activation layers, storing in `of_layers`.
  Calculate the running average of std for activation layers, storing in `std_layers`.
end for
Sort `of_layers` by ratio, `std_layers` by std. i ← 0
while `of_layers[i].ratio` ≠ 0 **do**
  Push `of_layers[i].layer` to `sens_list`, i ← i + 1
end while
for i = 0 **to** size(`std_layers`) - 1 **do**
  if `std_layers[i].layer` is not in `sens_list` **then**
    Push `std_layers[i].layer` to `sens_list`.
  end if
end for
return `sens_list`

5) Surpassing Full-Precision: To outperform full-precision, we introduce two additional techniques:

(1) Prompt-aware quantization: conditional diffusion models often employ classifier-free guidance [16], which requires the model to process two inputs: one with the given prompt and another with a null prompt. We propose to quantize only the path associated with the input prompt to low-precision, while maintaining the null prompt path in high-precision. This selective approach adds controlled noise that can improve the quality of the output image.

(2) Timestep-aware quantization: We quantize insensitive layers to low precision during the early timesteps (first 80%) and maintain high precision in the final steps. This helps the model recover from early noise, producing sharper images, as final timestep precision is crucial to avoid blurriness.

By following the five steps outlined in this section, we achieve our proposed FP-MP design, which not only approaches the quality of full-precision but also exceeds it. To summarize, only the portion of the activation tensors that receives the prompt during early timesteps for insensitive layers are quantized to FP8 (E4M3), while the rest remain

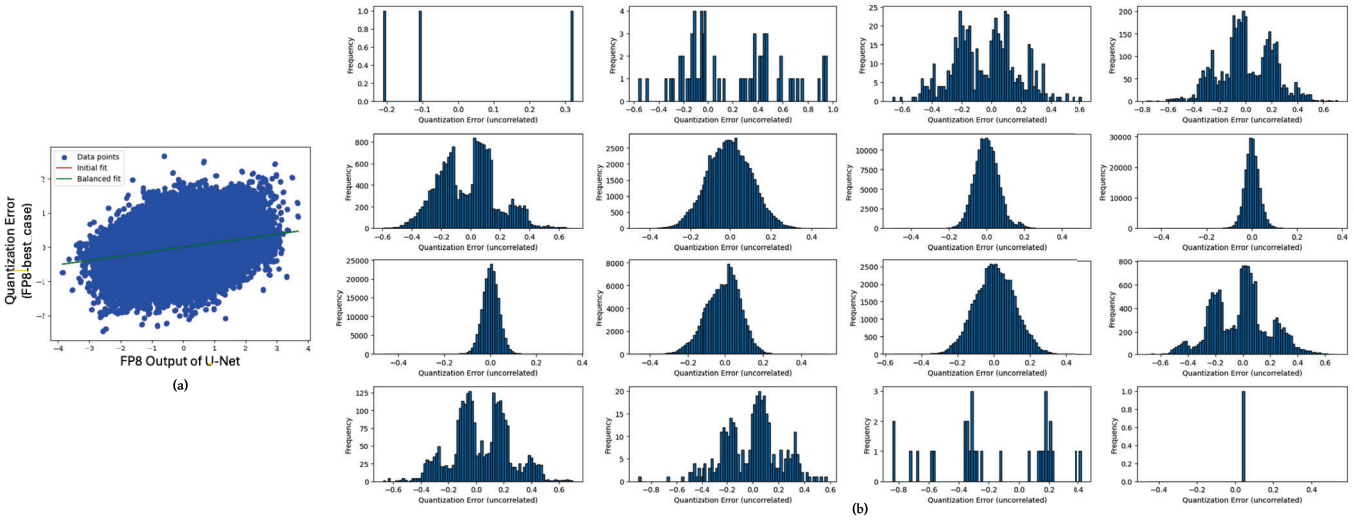


Fig. 3. (a) U-Net output quantization noise at timestep=12 for Stable Diffusion model using MS-COCO dataset: it is modeled with a correlated (linear) and an uncorrelated (nonlinear) component. (b) Distribution of the uncorrelated component for different ranges of U-Net output.

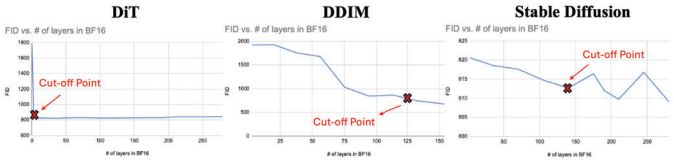


Fig. 4. Selecting cut-off point between high and low precision for different models: X-axis is the number of layers quantized to high-precision (the rest of the layers are quantized to low-precision) and Y-axis is FID (lower is better).

in BF16. For the weights, all sensitive (insensitive) layers are quantized to BF16 (FP8).

B. Noise Modeling

The goal of noise-tuning is to reconstruct FP-MP using low precision (FP8) with adjustable noise. We aim to correlate the U-Net low-precision output with the quantization error from transitioning to unified low precision, minimizing added parameters. Previous work [10] showed a linear correlation between the error and U-Net output for INT8 quantization.

We profile FP quantization error using the Stable Diffusion model and MS-COCO dataset. Fig. 3 (a) illustrates a *nonlinear* relationship between U-Net output in low-precision and quantization error, unlike integer quantization. The error comprises a correlated (linear) and an uncorrelated component (distance from the line). Fig. 3 (b) shows the uncorrelated component's distribution across U-Net output ranges, modeled by three overlapping Gaussians. Thus, we parameterize each Gaussian with trainable tensors that represent the mean, variance, and scaling of each distribution. We formulate the quantization noise as:

$$\mathbf{x}_{t-1,MP} = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_{t,MP} - \frac{\beta_t}{\sqrt{\alpha_t(1-\bar{\alpha}_t)}}\epsilon_{\theta,LP}(\mathbf{x}_{t,MP}, t) - \frac{\beta_t}{\sqrt{\alpha_t(1-\bar{\alpha}_t)}}\Delta\epsilon_\theta(\mathbf{x}_{t,MP}, t) + \sigma_t\mathbf{z} \tag{6}$$

$$\Delta\epsilon_\theta(\mathbf{x}_{t,MP}, t) = P_{t,0}\cdot\epsilon_{\theta,LP}(\mathbf{x}_{t,MP}, t) + \sum_{i=0}^{2} P_{t,1+3\cdot i}\cdot E$$
$$\text{where } E = \exp\left(-\frac{1}{2}\left(\frac{P_{t,2+3\cdot i} - \epsilon_{\theta,LP}(\mathbf{x}_{t,MP}, t)}{P_{t,3+3\cdot i}}\right)^2\right) \tag{7}$$

where $\mathbf{x}_{t,MP}$, $\mathbf{z}$, $\epsilon_{\theta,LP}$, $\Delta\epsilon_\theta$ represent data sample at timestep t with FP-MP, a sample from distribution $\mathcal{N}(0,\mathbf{I})$, U-Net output in low-precision, and U-Net output quantization error (low-precision vs. FP-MP). α_t, β_t, and σ_t are hyperparamters. $P_{t,j} \in \mathbb{R}^{C\times H\times W}$ ($j \in \{0,1,\ldots,9\}$) is j^{th} trainable parameter at timestep t.

C. Data Generation

We generate training data for noise-tuning by curating input prompts. For conditional models, prompts are sampled from the dataset; The FP-MP design outperforms full-precision for conditional models, so we use FP-MP U-Net output as ground truth. For unconditional models, full-precision U-Net output is used, as FP-MP is less effective without prompts.

D. Noise-Tuning

The noise-tuning phase automates fine-tuning of U-Net head noise parameters using data-driven learning with ground truth from Sec. IV-C. Only the new parameters are trained, keeping the rest of the model frozen to minimize overfitting and reduce computational load. Training runs for a set number of epochs, halting if improvements plateau.

E. Runtime Optimization

The nonlinear function in noise-tuning is memory-bound, limiting DM-Tune's speed. Recent lower-bit quantization exacerbates this issue. To address this, we optimize a GPU kernel by fusing low-precision matrix multiplication with Gaussian terms, implementing FP8 matrix multiplication using CUTLASS[1] and high-throughput tensor cores.

[1]https://github.com/NVIDIA/cutlass

TABLE I
COMPARING DM-TUNE PERFORMANCE WITH DIFFERENT FORMATS FOR STABLE DIFFUSION, DIT, AND DDIM MODELS. NT IS NOISE-TUNING.

Model/Dataset	Method	FID ↓	Precision ↑	Recall ↑	Authenticity ↑	CLIP ↑	Density ↑	Coverage ↑	KD ↓	SW-approx ↓
Stable Diffusion (MS COCO 512×512)	FP32 (Baseline)	760.9	0.92	0.87	62.79	31.44	0.91	0.86	0.077	0.15
	FP8	776.44	0.92	0.86	67.18	31.56	0.91	0.88	0.091	0.16
	INT8	3993.55	0.0	0.0	100.0	19.16	0.0	0.0	27.62	1.73
	Q-Diffusion (W8A8)	757.51	0.92	0.87	63.14	31.40	0.88	0.86	0.077	0.15
	Q-Diffusion (W4A8)	768.90	0.92	0.83	63.57	31.27	0.92	0.87	0.085	0.16
	Ours (w/o NT)	**741.51**	**0.94**	0.86	60.94	**31.58**	**0.95**	**0.89**	**0.070**	**0.14**
	Ours (w/ NT)	**737.89**	0.91	**0.89**	**64.31**	31.32	**0.94**	**0.92**	**0.075**	**0.14**
DiT (ImageNet 256×256)	FP32 (Baseline)	721.58	0.99	0.97	30.78	30.10	0.89	0.98	0.045	0.14
	FP8	1626.56	0.98	0.31	70.41	24.31	0.75	0.91	1.88	0.63
	INT8	4300.40	0.0	0.0	100.0	19.16	0.0	0.0	32.1	1.91
	Ours (w/o NT)	**704.74**	0.99	0.97	**33.77**	29.91	**0.94**	**0.99**	0.045	**0.13**
	Ours (w/ NT)	**701.23**	0.98	0.97	**32.94**	**30.87**	**0.99**	0.96	**0.041**	**0.12**
DDIM (CelebAHQ 256×256)	FP32 (Baseline)	659.35	0.85	0.39	89.55	N/A	0.71	0.55	2.16	0.48
	FP8	1971.28	0.28	0.02	98.24	N/A	0.08	0.05	8.16	1.02
	INT8	4379.36	0.0	0.0	100.0	N/A	0.0	0.0	40.0	2.07
	Ours (w/o NT)	750.16	0.80	0.29	**91.02**	N/A	0.54	0.45	2.41	0.51
	Ours (w/ NT)	**653.98**	0.82	**0.44**	**90.14**	N/A	0.66	**0.58**	2.18	**0.45**

The noise-tuning head performs matrix multiplication, casts outputs to full-precision, adds Gaussian terms, and casts back to FP8 using intrinsic functions. We apply three optimizations to further improve the performance: **O1**: Replace (7) with lookup tables (LUTs) in shared memory. Based on the profiling results (i.e., Fig. 3 (a)), we set a range for the values of $\epsilon_{\theta,LP}(\mathbf{x}_{t,MP}, t)$ and allocate 32 KB shared memory. We calculate the LUT index based on the value of $\epsilon_{\theta,LP}(\mathbf{x}_{t,MP}, t)$. **O2**: Overlap exponential calculations with data loading to enhance efficiency. We use rolling prefetch; prefetching one element and calculating exponentials for another element. **O3**: Use vector instructions to improve instruction-level parallelism and throughput. We found that grouping eight FP8 elements leads to the highest throughput.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

Models and datasets: We use three different types of diffusion models: Stable Diffusion v1.5 [17] (text-conditioned), Diffusion Transformer (DiT) [18] as a class-conditioned model, and DDIM [19] that is an unconditional model. The datasets are MS COCO [20], ImageNet [21], and CelebA-HQ (loaded from Hugging Face[2]), respectively. The resolution of the images in the first model is 512×512 and it is 256×256 for the other two models. All experimental settings (e.g., variance schedule, guidance scale, scheduler) follow the official implementation, with 50 time steps. We also use LDM [17] and IDDPM [22] for comparison with related work.

Quantization baselines: We compare the performance of our approach against these state-of-the-art: Q-Diffusion [8], PTQ4DM [9], and PTQD [10].

Implementation: We implement DM-Tune using PyTorch with model-agnostic quantization achieved via hook functions registered conditionally to specific layers, timesteps, and tensor portions. During the forward pass, activations are quantized and de-quantized, while the backward pass employs straight-through estimation (STE) [23]. Batch normalizations and activation functions remain in high precision. Models run on an A100 GPU (40 GB) for noise-tuning and quality evaluation, and on an L4 GPU (24 GB) for runtime assessment due to FP8 support. Diffusion models are evaluated with DGM-Eval [14].

[2]https://huggingface.co/google/ddpm-celebahq-256

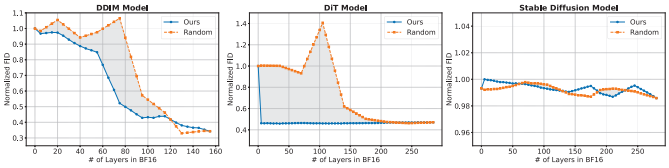


Fig. 5. Comparison of FID between our sensitivity-based quantization strategy and random precision selection, gradually shifting from entirely low-precision to entirely high-precision in the layers.

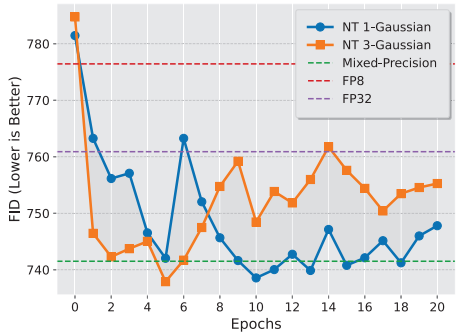


Fig. 6. Comparison of FID scores across epochs for two noise-tuning methods with one and three Gaussian terms for Stable Diffusion model using MS COCO evaluation dataset.

Noise-tuning uses the Adam optimizer with a learning rate of *1e-3*, 4K training samples, 1K evaluation samples, and batch sizes of 8 (Stable Diffusion), 64 (DiT), and 32 (DDIM), halved during noise-tuning to avoid out-of-memory (OoM) errors.

B. Mixed-Precision Sensitivity

We first demonstrate the effectiveness of our mixed-precision sensitivity analysis before applying prompt-aware and timestep-aware quantization. Sensitivity criteria are evaluated against a random baseline, which assigns high- and low-precision layers randomly based on a fixed split. Fig. 5 compares FID scores for three models. For DiT, our method effectively identifies sensitive layers, achieving a larger performance gap over the random approach. The gap is smaller for DDIM, while for Stable Diffusion, layer precision choices have minimal impact on accuracy.

C. Training

We evaluate noise-tuning performance across training epochs, comparing it to the FP-MP ground truth, FP8, and FP32 to assess convergence. To demonstrate its superior expressivity, we also compare noise-tuning with one Gaussian term (*NT 1_Gaussian*) to three terms (*NT 3_Gaussian*). Fig. 6 presents FID metrics during training on the MS COCO dataset

TABLE II
COMPARING DM-TUNE PERFORMANCE WITH STATE-OF-THE-ART. NT IS NOISE-TUNING.

Model/Dataset	Method	FID ↓	Precision ↑	Recall ↑	Density ↑	Coverage ↑	KD ↓	SW-approx ↓
IDDPM (ImageNet 64×64)	FP32 (Baseline)	282.02	0.80	0.84	0.77	0.81	0.15	0.20
	FP8	519.50	0.73	0.56	0.68	0.48	0.66	0.36
	INT8	4186.98	0.0	0.0	0.0	0.0	31.23	1.84
	PTQ4DM	376.09	0.78	0.80	0.78	0.72	0.29	0.27
	Ours (w/o NT)	**256.17**	**0.82**	**0.87**	0.74	**0.84**	**0.14**	**0.16**
LDM (ImageNet 64×64)	FP32 (Baseline)	265.83	0.98	0.42	0.98	0.89	0.059	0.12
	FP8	346.77	0.95	0.41	0.99	0.87	0.28	0.26
	INT8	3519.99	0.45	0.0	0.11	0.0	19.35	1.52
	PTQD	226.51	0.98	0.53	0.97	0.91	0.041	0.10
	Ours (w/o NT)	**232.79**	0.96	**0.51**	0.98	**0.93**	**0.048**	**0.09**

TABLE III
COMPARISON OF NOISE-TUNING OPTIMIZATION TECHNIQUES FOR MATRIX MULTIPLICATION (1K×1K MATRICES). NORMALIZED RUNTIME IS SHOWN.

FP8	NT (8 bit) w/o opt	NT (8 bit) O1	NT (8 bit) O1 + O2	NT (8 bit) O1 + O2 + O3
1.00X	1.12X	1.04X	1.02X	1.01X

with the Stable Diffusion model. *NT 3_Gaussian* converges faster and even surpasses the FP-MP ground truth. Notably, FP-MP performance is achieved in just 5-10 epochs, underscoring the efficiency and low training cost of our approach.

D. Image Generation Performance

We evaluate DM-Tune's performance against other quantization methods. Table I compares results for three diffusion models using 1K samples and DGM-Eval metrics. For Stable Diffusion and DiT, DM-Tune without noise-tuning (NT) outperforms full precision, with noise-tuning further improving results. For the unconditional DDIM model, DM-Tune surpasses the FP32 baseline only with noise-tuning, likely due to initial mixed-precision noise and subsequent fine-tuning with nonlinear parameters. Overall, DM-Tune improves quality, diversity, prompt alignment, and originality compared to full precision, especially for conditional models. It outperforms Q-Diffusion in most cases, as their integer-based quantization limits dynamic range and lacks techniques to exceed full-precision quality. Fig. 1(a) highlights I-MP's challenges with high-fidelity image generation for human faces.

We use the DINOv2-ViT model as the feature extractor to ensure fairness in evaluation, leading to higher FID results than those in related work. To address this, we re-evaluate related work using our method and feature extractor. Since prior methods support limited models and datasets due to their calibration techniques, we compare DM-Tune with them on the datasets they support. Table II shows a comparison using 5K samples with DGM-Eval metrics. DM-Tune outperforms prior methods in most scenarios, though PTQD achieves better results on some metrics.

E. Runtime

We evaluate the runtime of the proposed noise-tuning, starting with a single matrix multiplication kernel with a NT head. Table III compares the runtime of various optimizations for a kernel fused with noise-tuning (three Gaussian terms) on 1k×1k matrices, averaged over 10 runs and normalized to FP8 without noise-tuning. With all optimizations, the runtime matches FP8, highlighting the efficiency of our approach. Fig. 7 compares the model runtime and performance against FP32, FP8, and state-of-the-art methods. Fully optimized noise-tuning adds minimal overhead to FP8 while delivering a $5.2\times$ average performance boost over prior art.

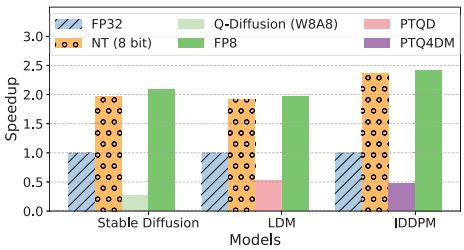


Fig. 7. Normalized model performance comparison of noise-tuning against state-of-the-art methods.

VI. RELATED WORK

Diffusion Model Quantization: Q-Diffusion [8] introduces post-training quantization (PTQ) using an 8/4-bit integer format with a timestep-aware calibration sampling mechanism. Similarly, PTQ4DM [9] applies 8-bit integer quantization but focuses on smaller models like DDPM and DDIM. To our knowledge, no prior work has demonstrated that quantization enhances image generation in diffusion models.

Noise Modeling: [10] introduces a mixed-precision integer quantization method that models quantization noise and applies corrections to reconstruct full-precision outputs. Unlike our approach, which uses mixed precision as ground truth, they assume full precision yields the best performance. Their reliance on statistical and manual profiling limits applicability to specific data formats and models.

VII. CONCLUSION

In this work, we introduced DM-Tune, a framework combining unified low-precision quantization with noise-tuning to enhance diffusion model efficiency and image quality. DM-Tune leverages inherent model noise to surpass the limitations of traditional mixed-precision methods, improving generation metrics. Experiments show that DM-Tune matches or outperforms full-precision models in quality, diversity, and text-to-image alignment while reducing inference time. Optimized GPU kernels further accelerate deployment. Future work will explore extending DM-Tune to lower-precision formats (FP4/6) and quantizing additional components like encoder-decoder structures to enhance runtime.

ACKNOWLEDGEMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, ComPort: Rigorous Testing Methods to Safeguard Software Porting, under Award Number 78284. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL01830. This work was also supported in part by NSF under Awards SHF-2326494.

REFERENCES

[1] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," 2021. [Online]. Available: https://arxiv.org/abs/2102.12092

[2] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi, "Photorealistic text-to-image diffusion models with deep language understanding," 2022. [Online]. Available: https://arxiv.org/abs/2205.11487

[3] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022. [Online]. Available: https://arxiv.org/abs/2112.10752

[4] Y. Liu, K. Zhang, Y. Li, Z. Yan, C. Gao, R. Chen, Z. Yuan, Y. Huang, H. Sun, J. Gao, L. He, and L. Sun, "Sora: A review on background, technology, limitations, and opportunities of large vision models," 2024. [Online]. Available: https://arxiv.org/abs/2402.17177

[5] C. Liu, C. Wu, S. Cao, M. Chen, J. C. Liang, A. Li, M. Huang, C. Ren, Y. N. Wu, D. Liu, and T. Geng, "Diff-pic: Revolutionizing particle-in-cell nuclear fusion simulation with diffusion models," in *The Thirteenth International Conference on Learning Representations*, 2025.

[6] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach, "Sdxl: Improving latent diffusion models for high-resolution image synthesis," 2023. [Online]. Available: https://arxiv.org/abs/2307.01952

[7] C. Liu, R. Song, C. Wu, P. Haghi, and T. Geng, "Instatrain: Adaptive training via ultra-fast natural annealing within dynamical systems," in *The Thirteenth International Conference on Learning Representations*, 2025.

[8] X. Li, Y. Liu, L. Lian, H. Yang, Z. Dong, D. Kang, S. Zhang, and K. Keutzer, "Q-diffusion: Quantizing diffusion models," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 17 489–17 499.

[9] Y. Shang, Z. Yuan, B. Xie, B. Wu, and Y. Yan, "Post-training quantization on diffusion models," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 1972–1981.

[10] Y. He, L. Liu, J. Liu, W. Wu, H. Zhou, and B. Zhuang, "Ptqd: accurate post-training quantization for diffusion models," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NeurIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2024.

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[12] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020. [Online]. Available: https://arxiv.org/abs/2006.11239

[13] A. M. Alaa, B. van Breugel, E. Saveliev, and M. van der Schaar, "How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models," 2022. [Online]. Available: https://arxiv.org/abs/2102.08921

[14] G. Stein, J. Cresswell, R. Hosseinzadeh, Y. Sui, B. Ross, V. Villecroze, Z. Liu, A. L. Caterini, E. Taylor, and G. Loaiza-Ganem, "Exposing flaws of generative model evaluation metrics and their unfair treatment of diffusion models," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[15] M. F. Naeem, S. J. Oh, Y. Uh, Y. Choi, and J. Yoo, "Reliable fidelity and diversity metrics for generative models," 2020. [Online]. Available: https://arxiv.org/abs/2002.09797

[16] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022. [Online]. Available: https://arxiv.org/abs/2207.12598

[17] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022. [Online]. Available: https://arxiv.org/abs/2112.10752

[18] W. Peebles and S. Xie, "Scalable diffusion models with transformers," 2023. [Online]. Available: https://arxiv.org/abs/2212.09748

[19] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=St1giarCHLP

[20] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015. [Online]. Available: https://arxiv.org/abs/1405.0312

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[22] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021. [Online]. Available: https://arxiv.org/abs/2102.09672

[23] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, "Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets," 2019.

SSDTrain: An Activation Offloading Framework to SSDs for Faster Large Language Model Training

Kun Wu*†‖, Jeongmin Brian Park*†‖, Xiaofan Zhang*‡,
Mert Hidayetoğlu§, Vikram Sharma Mailthody†, Sitao Huang¶, Steve Lumetta‖, Wen-mei Hwu†‖
†NVIDIA ‡Google §Stanford University ¶University of California, Irvine ‖University of Illinois Urbana-Champaign
*These authors made equal contributions

***Abstract*—The growth rate of the GPU memory capacity has not been able to keep up with that of the size of large language models (LLMs), hindering the model training process. In particular, activations—the intermediate tensors produced during forward propagation and reused in backward propagation—dominate the GPU memory use. This leads to high training overheads such as expensive weight update costs due to the small micro-batch size. To address this challenge, we propose SSDTrain, an adaptive activation offloading framework to high-capacity NVMe SSDs. SSDTrain reduces GPU memory usage without impacting performance by fully overlapping data transfers with computation. SSDTrain is compatible with popular deep learning frameworks like PyTorch, Megatron, and DeepSpeed, and it employs techniques such as tensor deduplication and forwarding to further enhance efficiency. We extensively experimented with popular LLMs like GPT, BERT, and T5. Results demonstrate that SSDTrain reduces 47% of the activation peak memory usage. At the same time, SSDTrain perfectly overlaps the I/O with the computation and incurs negligible overhead. Compared with keeping activations in GPU memory and layerwise full recomputation, SSDTrain achieves the best memory savings with negligible throughput loss. We further analyze how the reduced activation memory use may be leveraged to increase throughput by increasing micro-batch size and reducing pipeline parallelism bubbles.**

I. Introduction

LLMs now drive a wide range of applications, including chatbots [1], search [2], content generation [3], reasoning [4], etc. These models, when sufficiently large in size, demonstrate emergent abilities [5] and thus the capability of handling complicated tasks. Such a phenomenon drives model designers to continue to scale up the size of LLMs, carrying more parameters. The already formidably high training costs continue to grow: training GPT-4, for example, cost US$100 million, a 21× increase over training GPT-3 [6].

GPU memory capacity has become a bottleneck for the continued growth of LLMs. As Fig. 1 shows, the increase of GPU memory capacity is around 60% slower than the LLM size scaling speed and the GPU FP16 throughput improvement. About 80% of the GPU memory used to train recent LLMs consists of activations [7], [8], the intermediate tensors produced by forward propagation and reused in backward propagation. Furthermore, the memory needed for activations is growing more rapidly than any other memory use, making GPU memory a more serious constraint for future LLM training (see Sec. II-B for details).

Common mitigations are to reduce batch size or use gradient accumulation. With gradient accumulation, a batch is divided into micro-batches that are processed separately between gradient updates. Although gradient accumulation has been adopted by many LLMs [9]–[11], the GPU computation stack is not designed for small inputs, and both mitigations lead to device under-utilization [12], [13] and suboptimal math library performance [14]. Intuitively, a smaller batch size might reduce total training computation through faster convergence. However, LLM trainers have identified a critical batch size to each model, below which convergence speed increases negligibly or even decreases [15], [16]. Notably, critical batch size grows during training, as training loss is reduced.

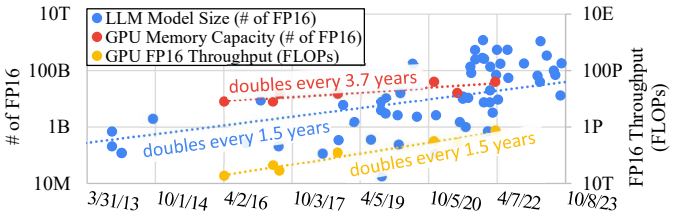


Fig. 1. The growth of FP16 throughput of GPUs for deep learning training is aligned with the model size of LLMs, but GPU memory capacity falls behind [18]. Horizontal axis shows release date. Points represent both Nvidia 100-level GPUs since K100 and Google TPUs.

Another common approach to reducing GPU memory use is activation checkpointing. With this, only some activations are kept in GPU memory, while others are flushed and then recomputed during backward propagation. For an L-layer model, activation checkpointing reduces memory requirements from $O(L)$ to $O(\sqrt{L})$ [17]. However, as Sec. II-B shows, even this alone is insufficient to eliminate the bottleneck posed by GPU memory limits for future LLMs.

This work proposes SSDTrain, a software framework that offloads activations to NVMe SSDs and reloads activations just before they are needed in backward propagation. SSDTrain is able to overlap activation transfers fully with computation, thereby reducing activation memory usage without incurring significant performance overhead. SSDs are a more attractive target than main (CPU) memory for several reasons. First, clusters and cloud instances [19]–[21] are typically limited in host memory capacity (100–250 GB/GPU), while SSDs offer much higher capacity. The limited host memory is also consumed by input, metadata, etc., which further reduces the amount of memory available for activation offloading. Second, host memory bandwidth is shared across training management tasks and offloaded computation [22]–[24] running on the host CPU and can be quite limited and even unpredictable [25] for saving and restoring activations. In contrast, the SSD bandwidth can be dedicated to the activation offloading during training. Third, SSDs are more elastic, both by adding more SSDs and even PCIe switches if necessary—as well as through the use of optional remote high-throughput storage [26], [27]. Such elasticity allows data centers to keep up with the fast-growing size of activations. In contrast, the memory capacity of GPU cloud instances and cluster nodes is much more difficult to extend.

This work makes the following main contributions.

1. We design and implement the SSDTrain framework to offload LLM activations to NVMe SSDs. We demonstrate the viability of SSDTrain on large-scale systems by modeling the performance,

The code repository is available at https://github.com/K-Wu/FlashTrain. Corresponding author: kunw@nvidia.com.

SSD lifespan and the required per-GPU PCIe bandwidth.
2. With all code in Python except for a tiny CUDA API hooking library, SSDTrain works with the latest PyTorch, and distributed frameworks. We developed and tested SSDTrain with Megatron-DeepSpeed [28] on a 2-GPU node with 7× Intel Optane SSDs.
3. Evaluation shows SSDTrain matches the original system's training time while reducing the activations peak memory use by up to 47%. This proves that SSDTrain overlaps the data transfer fully with computation. Compared with keeping activations and layer-wise full recomputation, SSDTrain obtains the best performance and the least memory peak. We further analyze how the reduced activation memory use may increase throughput by increasing micro-batch size and reducing pipeline parallelism bubbles.

II. Background

A. Transformer-Based LLM

Most LLM architectures, including GPT [29], are transformer-based [30]. These models consist mainly of transformer layers. Each transformer layer is primarily made up of an attention block and a multi-layer perceptron (MLP) block. GPT is a decoder-only model because it only involves transformer decoder layers. An encoder layer has the same structure as the decoder layer except that the latter imposes causality on the attention mask. Transformer models are classified as (1) encoder-only, e.g., BERT [31], (2) decoder-only, and (3) encoder-decoder, e.g., T5 [32]. In encoder-decoder models, decoder layers take in both outputs from the encoders and another text and apply two attention blocks—the self-attention block is applied to the new text, and the cross-attention block is applied among the tokens in the sequence from the encoder and tokens in the new text.

Parallelizing LLM training involves partitioning and/or replicating the model and the data into different GPUs [33]. Pipeline parallelism (PP), data parallelism (DP), and tensor parallelism (TP) are the three widely-adopted levels of parallelism available to all LLM models. PP divides the model and places chunks of layers on different GPUs. In a step, when the GPUs finish their layers, the output is passed to the GPUs owning next layers. DP replicates the models in different groups of GPUs and assigns separate micro-batches to each group. TP shards a weight tensor and puts shards onto different GPUs. Each GPU performs a portion of the computation using its shard for the corresponding operator. Zero Redundancy Optimizer (ZeRO) [34] further reduces memory use with DP by sharding the optimizer states, and/or optionally the gradients and parameters across these GPUs.

B. GPU Memory Capacity and Model Throughput

As Fig. 7 of Sec. IV will show, the GPU memory capacity is limiting the model throughput. By offloading the activations to SSDs, SSDTrain can alleviate this limitation and improve the per-GPU model throughput. An important question is if the GPU memory capacity will continue to be the limiting factor of per-GPU model throughput according to the trend of LLM scaling. This section shows that the historical trend will make GPU memory capacity an even more important limiting factor of the per-GPU model throughput.

Neural scaling laws [15], [16], [35] guide LLM scaling as computing power increases. We follow these laws in our reasoning. The whole-system GPU compute throughput $C \propto ND_{batch}$, where N is the number of parameters and D_{batch} is the number of tokens in a batch [36]. Chinchilla Scaling [35] concludes that the optimal model design follows $N \propto C^{0.5}$, which implies $D_{batch} \propto C^{0.5}$ to saturate the GPU throughput. Whole-system GPU memory use consists of two parts: activations, which require $S_{activations} \propto \frac{N}{h} D_{batch}$,

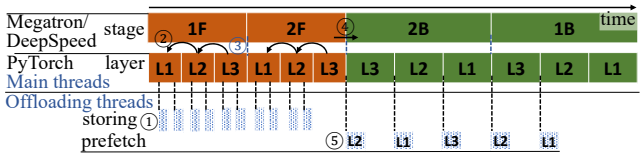


Fig. 2. SSDTrain timeline of a step of a 2-microbatch 3-layer (L) model.

where h is the hidden dimension in the layers and is a slow growing function of N, e.g., $h \propto N^{1/3}$, and all other memory use, $S_{others} \propto N$, including parameters, gradients, and optimizer states. Comparing the factors, we can deduce that (1) $S_{activations}$ grows faster than S_{others}, and (2) whole-system memory use, which is dominated by the activations, grows slightly slower than the compute throughput C (approximated $C^{5/6}$). However, Fig. 1 shows that the GPU memory capacity historically grows (red dotted line) at 41% the growth rate of the compute throughput (yellow dotted line). Therefore, **GPU memory capacity will become increasingly inadequate for saturating the compute throughput, and memory for activations will continue to dominate the GPU memory usage.**

What about activation checkpointing? Revisiting the prior equation, $S_{activations} \propto \frac{N}{h} D_{batch} \propto LhD_{batch}$ where L is the number of layers. Checkpointing makes the new activations memory use $S'_{activations} \propto \sqrt{L}hD_{batch}$. Since L and h grow when N increases and $D_{batch} \propto C^{0.5}$, $S'_{activations}$ still grows faster than S_{others}.

C. SSD Endurance

Trends in price, latency, and bandwidth have led to the widespread adoption and integration of SSDs into cloud instances and clusters [19]–[21]. Flash's random write latency is reduced to tens of microseconds [37], and NVMe SSD data rates are now a few GB/s.

SSD endurance remains a concern: how long will SSDs last in the write-intensive activation offloading? SSD endurance is determined by the type and number of cells, write amplification factor (WAF), and over-provisioning. SSD cells can be purposed to store one bit or multiple levels. Generally, the more bits a cell stores, the shorter its lifetime in program-erase (PE) cycles. WAF is the ratio of media write amount to host write amount—SSD writes pages at a time but erases blocks of pages, a coarser granularity. Erasing a partially empty block requires that remaining valid pages be relocated, causing write amplification. In turn, vendors adopt over-provisioning to reserve some blocks for wear leveling, evening out the writes across blocks.

Notably, SSD endurance rating uses the JESD testing method [38] which performs random writes after tough preconditioning. In our scenario, the writes are large and sequential as each tensor being offloaded is easily hundreds of MBs in size. Such writes are more endurance-friendly compared with the writes used to determine the JESD rating. For example, 3-DWPD SSDs generally allow about 2.5× as many sequential writes than expected from the JESD rating [39]–[41]. Vendor guidelines [42]–[44] and empirical data [45] corroborate this difference. Sec. III-D uses modeling to demonstrate why mainstream data center SSDs are viable options to support SSDTrain deployment in a large-scale LLM training system.

D. SSD Offloading Systems for LLM

GPUDirect Storage (GDS) enables a direct data path between GPU and NVMe SSDs [48]–[50], removing the need for a CPU bounce buffer to enhance bandwidth and reduce both latency and CPU load.

To mitigate the training overhead caused by the GPU memory capacity limit, SSDTrain has three key differences with related work [50], [51]: SSDTrain offloads (a) activations to (b) the SSDs

TABLE I
COMPARING SSDTRAIN TO OTHER LLM SYSTEMS WITH ACTIVATION OFFLOADING FEATURES [46]–[48]. WITHOUT BACKWARD PROPAGATION, INFERENCE SYSTEMS MAY DISCARD INTERMEDIATE TENSORS ONCE A LAYER IS DONE. WE GENERALIZE "**ACTIVATION**" TO REFER TO KEY-VALUE (KV) CACHE AS WELL BECAUSE IT IS REUSED ACROSS STEPS.

		Flexgen	LLM in a Flash	ZeRO-Infinity	**SSDTrain**
Training				✓	✓
Activation offloading	**to main memory**	✓	✓	Checkpoints only	✓
	to SSD	✓			✓
Direct GPU–SSD data path					✓
Async data transfer					✓
Interoperability					✓

(c) with negligible performance overhead. To the best of our knowledge, SSDTrain is the first work that leverages SSD to offload activations for LLM training. Table I illustrates some other SSDTrain's features:

Direct and async GPU–SSD data transfer. As Sec. I shows, transfer via CPU impacts efficiency. Besides, existing systems either block the training when loading offloaded data, or synchronize at each layer. Thus, the I/O latency is in the critical path. SSDTrain hides the I/O latency by overlapping I/O with computation.

Interoperability. Since LLM training requires a synergy of Python packages and the ecosystem is rapidly evolving, it is vital for offloading to have good interoperability with other components. SSDTrain logic is local to processes and can work with distributed frameworks, e.g., Megatron. In contrast, DeepSpeed's offloading features, e.g., ZeRO-Infinity, are available only in certain ZeRO stages. ZeRO stage determines what is sharded. For example, stage-3 ZeRO in Fig. 5 shards optimizer states, gradients, and weights across the GPUs.

III. DESIGN AND IMPLEMENTATION

A. Overview of the SSDTrain Framework

SSDTrain implements a *tensor cache* to manage the offloading and reloading of tensors, facilitating the release of memory as well as the prefetch of tensors back to memory before they are needed for backward propagation. Fig. 2 exemplifies how SSDTrain works. SSDTrain launches its own threads to store tensors (①) and to reload them (⑤). In forward propagation (F), offloading of an activation starts once the operator producing it finishes (①). When activations are reused in backward propagation (B), prefetching (⑤) occurs in the reverse order of layers as recorded during forward propagation (②). If the last layer begins backward propagation immediately after its forward propagation (L3 in micro-batch 2 in the example), its activations are kept (④). SSDTrain keeps individual records for each micro-batch. Upon micro-batch changes (②), SSDTrain switches its own record to the one corresponding to the new micro-batch.

Fig. 3 shows the SSDTrain workflow. SSDTrain retrieves the amount of computation and activation size of the model from the model instance, GPU throughput, and SSD bandwidth. Then, SSDTrain sets the activation offload amount accordingly. The tensor cache manages the activations and performs tensor offloading and loading. To achieve this, it uses PyTorch hooks to alter PyTorch execution. Sec. III-B details the design and implementation of the tensor cache. SSDTrain has the SSD offloader that targets NVMe SSDs within the same node and the CPU offloader that targets host memory. Each offloader encapsulates the logic to transfer CUDA tensors to and from a target. The SSD offloader leverages the GDS python binding, kvikio [52]. Using the `LD_PRELOAD` interposition mechanism, the CUDA malloc hook library alters CUDA memory allocation and free API calls so that the memory is properly registered and deregistered for best GDS performance. This allows us to keep the PyTorch memory allocator for easy comparison with the baseline, without replicating its implementation in a PyTorch pluggable memory allocator or modifying the PyTorch C++ code. The CPU offloader is for future work on clusters with massive remote SSD storage. It is backed by an allocator with pre-allocated host-pinned memory. The pool size is determined by profiling the first training step. Hints are added to Megatron's and DeepSpeed's schedulers, e.g., ③ and ④ in Fig. 2. E.g., for DeepSpeed's scheduler, hints are added before and after the execution of each command, e.g., computing the micro-batch `i`, communication so that the tensor cache gets notified about the upcoming stage and the completion of an action. Accordingly, the tensor cache can prefetch data, or wait for I/O to complete.

To use SSDTrain, a few lines are to be added in the existing script: They register the PyTorch hooks, bookkeep the weights to not offload them, and monkey-patch [53] the schedulers.

SSDTrain extends naturally to distributed settings such as use with ZeRO, because frameworks such as DeepSpeed and Megatron divide the workload into processes built on top of PyTorch's built-in tensor functionality. By working below PyTorch and keeping each process's activities local, SSDTrain applies directly to distributed launches.

B. Hook-Based Implementation of Tensor Cache

To benefit from tensor offloading, the GPU memory that the offloaded tensors own must be released when the tensors are not in use. However, PyTorch by default stores a reference to all the activations on the computational graph, disallowing the GPU memory to be reclaimed. The tensor cache alters the PyTorch execution so that the identifiers of the activations are registered on the computation graph; upon PyTorch's reusing the activation tensor, the tensor cache uses the identifier from the computational graph as the key to return the requested tensor. In the forward propagation, when the tensor finishes offloading, the tensor cache no longer holds a reference to it, allowing its memory to be reclaimed by Python garbage collection once the control flow gets out of the function scope where the tensor object is used. In the backward propagation, the tensor cache holds a reference to the tensor by loading it from the SSD before its use; when all the module scopes the tensor is referred to have been finished, the reference is no longer held, allowing its memory to be reclaimed. In short, the tensor cache is the in-memory structure that manages the references to all activations and tracks activations' states, including if they are being offloaded, the path in the file system, etc.

Tensor cache uses PyTorch hooks to alter its execution behavior. The forward hook pair works in the forward propagation: The start of a module triggers the forward pre hook, and the finish of a module triggers the forward hook. Tensor cache maintains the current scope stack using the forward hook pair: Upon entrance to a module, the module is pushed to the stack; Upon module exit, it is popped out.

Backward hook pair is similar. When entering a module, the tensor cache prefetches activations in upcoming modules. Sec. III-C2 details prefetching. When exiting a module, the tensor cache removes it from the scope lists of all activations. Activations no longer in use are removed, whose memory will be released by garbage collection.

When a tensor is to be registered onto the computation graph, the pack hook is called to produce a value to be registered instead. When the tensor is reused, the unpack hook is called to take in the object on the computation graph and return the original tensor.

Fig. 4 illustrates tensor cache's activity when pack or unpack hook is triggered. When the multiply operator $x \cdot w$ finishes (①), the pack hook is called (②) on the input `x` and weights `w`. Tensor cache has a record of weights, and accordingly returns `w` to let it be registered on the graph as is. The tensor will also be returned as is if the tensor is on CPU or it is too small (Line 2 in Alg. 1). As line 6 in Alg. 1 shows, the tensor cache does not offload tensors but only keeps a record when the module is to be kept in the memory or in backward propagation. The first condition holds true when the amount of activation in this step before the current tensor reaches the size set in Fig. 3. The second condition is true when an activation-checkpointing-enabled function does recomputation in the backward propagation to reproduce the activations. For tensor `x` in Fig. 4, the tensor cache stores it to the SSDs (③), updates the amount of activations offloaded in this step, and returns a tensor identifier. When the unpack hook is triggered (Ⓑ), in the backward propagation (Ⓐ), the tensor cache either waits until the prefetch finishes(Ⓒ), and eventually returns the tensor.

C. Tensor Cache Mechanisms and Optimization

1) Deduplicating Tensors and Excluding Weights: Tensor cache has a `get_id()` function to assign a unique identifier to each tensor. The shortcoming of PyTorch native `id()` is that its returned value is related to the GPU memory address. As SSDTrain offloads activations, the latter will be cleared by garbage collection once the control flow goes out of its use scope. The GPU memory address may be reused, causing identifier collision. To solve this, `get_id()` combines the timestamp when it first processes the tensor with the tensor shape as the unique identifier: When `get_id()` processes a tensor `t` for the first time, `get_id()` adds the current timestamp as an additional attribute to the tensor's underlying storage `t.untyped_storage()` instead of `t`. This is because sometimes PyTorch creates new `torch.Tensor` objects representing the identical tensor. All future `get_id()` calls get the attribute value. This deduplicating scheme helps prevent redundant I/Os.

PyTorch registers all needed tensors in backward propagation into the computational graph, involving activations and weights. As this work focuses on activations, the tensor cache excludes the weights. To this end, before training, the tensor cache records the identifiers of all weights. As linear layers store the transpose of weights for backward propagation, the unique identifiers of the transpose are recorded. One benefit of our `get_id()` scheme is that the identifier for the transpose of the same weights tensor remains consistent across steps. This is because the transpose uses the original tensor's underlying storage, which we already assigned a timestamp to before training.

2) Offloading and Forwarding Tensors: The tensor cache has two thread pools—one for storing tensors and the other for loading tensors. Submitted jobs are executed in first-in-first-out (FIFO) order.

To hide the I/O latency, the tensor cache starts prefetching each activation before the corresponding module's backward propagation. The activations in the last module are kept in GPU memory so they need not be prefetched. This simple scheme suffices because in PyTorch, CPU submits GPU kernel launches and memory operations ahead of GPU execution. Prefetching schemes are equivalent as long as there are always I/O tasks in GPU job queue to keep PCIe busy.

Upon loading a tensor, if it is still being stored, the tensor cache will return its in-memory reference to skip loading from SSD. We call this data forwarding. E.g., in Fig. 4, when PyTorch retrieves `x` from the `MulBWD` node, if `x` is still being stored, it is in memory. Instead of loading the tensor, the tensor cache returns `x`'s in-memory reference converted from the weak reference and store the obtained reference in the tensor cache for future if it is used in other scopes.

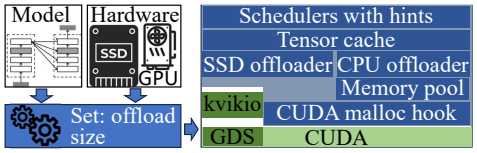


Fig. 3. SSDTrain workflow. SSDTrain components are shown as blue blocks.

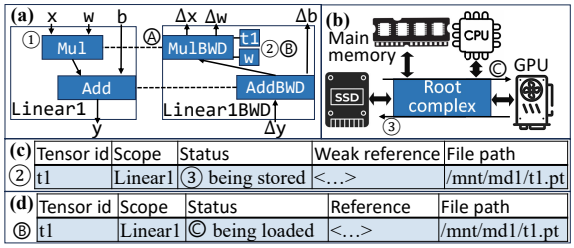


Fig. 4. Tensor cache registers hooks to offload tensors and reload tensors. **(a)** shows the computational graph. **(b)** shows the hardware data path. **(c)** and **(d)** show the tensor cache state when the pack or unpack hook is triggered.

Algorithm 1: Pack–unpack hook pair used by the tensor cache.

Input: Tensor cache `tc`, current scope `module`, tensor to pack `t`, and/or object to unpack `obj`.

```
1  Function pack_hook(t):
2      if tc.is_weights(t) or t.is_cpu or
         math.prod(t.size())<2**20 then return t
3      tid = get_id(t)
4      tc.add_to_current_scope(tid)
5      if tc.is_offload_amount_reached() or
         tc.is_current_in_backward() then
6          tc.keep_in_gpu_memory(tid,t)
7      else tc.offload(tid,t)
8      return tid
9  Function unpack_hook(obj):
10     if isinstance(obj,torch.Tensor) then return obj
11     tc.load_or_wait_load(obj)
12     return tc.get_loaded_tensor(obj)
```

D. SSD Write Amount, Bandwidth, and Lifespan

To confirm if our design is viable in large-scale systems, particularly concerning SSD endurance and required bandwidth, we conduct performance modeling to obtain forward propagation time per training step and the size of activations produced in the process. We extend the performance model package `llm-analysis` [54], which models the forward propagation of each transformer layer as a simple pipeline: $t = \max\left(\sum_l \max\left(t_{l,compute}, t_{l,memory}\right), t_{ZeRO,communicate}\right)$, where l denotes any layers inside a transformer layer. When ZeRO is enabled, the ZeRO communication time is assumed to be perfectly pipelined with the non-ZeRO operations at the transformer layer level.

We model the required PCIe write bandwidth per GPU as the total amount of activations divided by half the training time. The lifespan is then projected as $t_{life} = S_{endurance} \cdot t_{step} / S_{activations}$ where $S_{endurance}$ is the lifetime writes allowed by the SSD endurance rating, $S_{activations}$ is the amount of activations per training step, and t_{step} is the step time. We validated the $S_{activations}$ formula with experiments in Sec. IV. We assume four Samsung 980 PRO 1TB for each GPU, and assume the WAF is 2.5 in JESD rating and 1 in our scenario. We also relax the data retention period: NAND flash gets 86× PE cycles when the data retention period is relaxed from 3 years

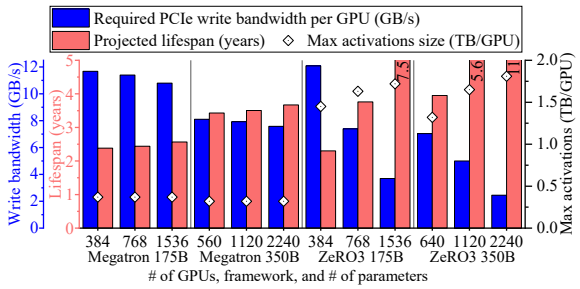


Fig. 5. Estimate of SSD lifespan, PCIe write bandwidth and maximal activations size per GPU. Lifespans longer than 5 years are shown on top of the pink bars. ZeRO3 stands for DeepSpeed with stage-3 ZeRO.

TABLE II
EVALUATION SYSTEM CONFIGURATION.

CPU (Memory)	2× AMD EPYC 7702 64-core (DDR4-3200 1TB)
GPU	2× Nvidia A100 40GB PCIe with NVLink
SSD	7× Intel Optane P5800X 1.6TB. 2× RAID0 arrays.
Software	Ubuntu 20.04.6 (5.15.0-113), CUDA 12.2 (driver 535.183.01), PyTorch 2.2.2, DeepSpeed 0.14.2, Megatron-DeepSpeed [28] (latest), kvikio 24.08

to 1 day [55]–[58]. With these, we obtain Fig. 5. We use measured data from Megatron-LM [10]. GPUs are A100 PCIe. Among all cases, the projected lifespan is more than 2 years, and the write bandwidth per GPU is no greater than 12.1GB/s. Moreover, when the system size and/or the model size scales up, the required PCIe write bandwidth reduces, and the projected lifespan increases. This is because larger systems imply increased communication overhead and reduced computation efficiency, slowing down training on GPUs.

We also estimate the maximal activations size each GPU produces per step by assuming only two layers in a row are in GPU memory at the same time while all other activations are offloaded. Then, diamond marks show the activation a largest micro-batch produces in a step in Fig. 5. The maximal activations size per GPU ranges from 0.4 TB to 1.8 TB, while the micro-batch size ranges from 8 to 32. Activations so large can no longer be held by the main memory and therefore SSD is the only choice as offloading target.

IV. EVALUATION

A. Experimental Setup

We use a machine with 2× A100 PCIe GPUs and 7× Intel P5800X SSDs, as Table II specifies. The SSDs are organized into two RAID0 arrays: one with 3 SSDs, and the other with 4 SSDs. Each array is the dedicated target of one A100. We measured the memory use of the A100 with 4 SSDs during evaluation. For consistency, the GPUs are locked at base frequency. The latest Megatron-DeepSpeed [28] is used, which incorporates DeepSpeed techniques into Megatron.

We measure the pretraining performance on BERT [31] as an encoder-only model, GPT [29] as a decoder-only model, and T5 [32] as an encoder-decoder model. We use the OSCAR dataset [59], [60].

We use the two GPUs for TP. The number of micro-batches per step is fixed at 1 because without PP, a new micro-batch will not start before both forward propagation and backward propagation of the previous micro-batch are done. More micro-batches only brings in gradient accumulation and does not affect the activation offloading pattern. In other words, unless stated otherwise, the micro-batch size is equivalent to batch size throughout Sec. IV. The hidden dimension is from 8192 to 16384, and we use typical hyperparameters [31], [32], [61] for this range. The attention head dimension is 128. The

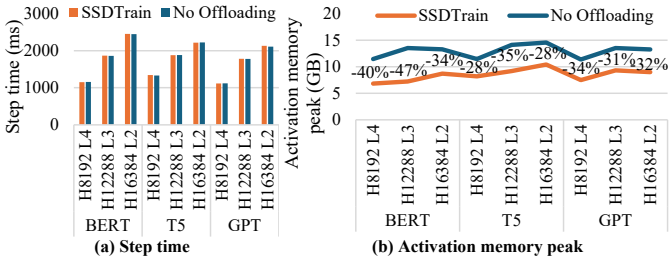


Fig. 6. Comparing SSDTrain to runs without tensor offloading. We test with different hidden dimensions (H) and number of layers (L). Batch size is 16.

text sequence length is 1024. For T5, the number of decoders is half of the total number of layers, rounded down. FlashAttention-2 [62] is used with or without SSDTrain for optimized attention computation.

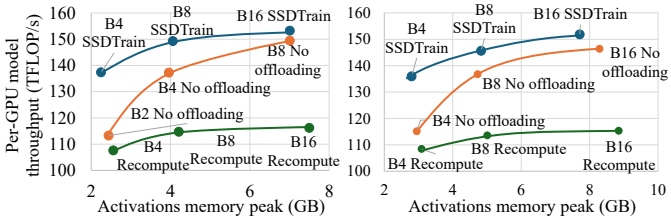
As each A100 has only 40GB of device memory, to explore the design space closer to that in real-world training systems with A100 80GB and later GPUs [7], [10], we make several mitigations. First, we use FP16 instead of mixed precision, eliminating the FP32 weight copy. Second, we use SGD instead of Adam as the optimizer to reduce optimizer states. The two measures only affect accumulation operations and weight updates, thus imposing a constant bias in step time and memory use in execution with or without SSDTrain.

B. Performance and Peak Memory Usage

To understand SSDTrain's impact on execution time and peak memory usage, we measure the step time of BERT, T5, and GPT and the memory peak during forward and backward propagation. The collected metrics of system with SSDTrain and without are compared in Fig. 6. For each model, we collected three scenarios with different (hidden dimension, number of layers): (8192, 4), (12288, 3) and (16384, 2). As shown, SSDTrain has almost no performance overhead in all cases. Although SSDTrain and its optimizations introduce additional CPU logic, the performance comparison indicates that this logic is not on the critical path. Rather, GPU computation defines the critical path, and the CPU's role lies primarily in launching new GPU jobs before current GPU operations complete. Thus, the CPU is underutilized, and SSDTrain's extra work does not lead to delay in new tasks reaching the GPUs. SSDTrain effectively reduces the activations' memory use peak by 28%–40% in these cases.

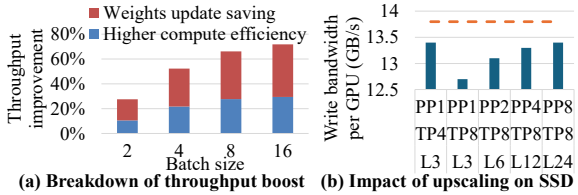
C. Comparing the Activations Placement Strategies

SSDTrain opens up offloading activations to SSDs as an option besides keeping activations in the GPU memory, and activations checkpointing. We compare the three different strategies here on the recompute-offload-keep (ROK) curve: Fig. 7 shows the training of two 3-layer BERT models. One set the hidden dimension as 12K and the other set it as 14K. In a ROK curve, each run is represented by a point. The x-axis is the activations memory peak. The y-axis is the model throughput [10], i.e., the number of algorithmic computations involved in the training step regardless of software and hardware implementation, e.g., whether the activations are recomputed, divided by the training step time. In these two cases, SSDTrain reduces the GPU activations memory peak, allowing for a larger batch size to attain higher throughput. Given the same batch size, SSDTrain offloading attains the throughput the same as the throughput when the activations are kept in memory. Meanwhile, SSDTrain gets a lower activations memory peak than the recomputation. Compared with keeping the activations in memory, SSDTrain is able to double the batch size with the same activations memory budget.



(a) H12288 L3 (b) H14336 L3

Fig. 7. Recompute-offload-keep (ROK) curve of 3-layer (L) BERT with hidden dimension (H) as 12K or 14K. Different batch sizes (B) are tested.



(a) Breakdown of throughput boost by larger micro-batch size (b) Impact of upscaling on SSD bandwidth

Fig. 8. Case study of 3-layer BERT with hidden dimension as 12K.

TABLE III
THE PER-GPU OFFLOADED TENSOR AMOUNT, ESTIMATE, AND REQUIRED PCIE WRITE BANDWIDTH OF BERT WITH DIFFERENT HIDDEN DIMENSIONS (H) AND NUMBER OF LAYERS (L). BATCH SIZE IS 16.

	H8192 L4	H12288 L3	H16384 L2
Offloaded amount	10.37 GB	12.85 GB	10.75 GB
Model estimate	11.13 GB	12.60 GB	11.50 GB
PCIe write bandwidth	18.0 GB/s	13.8 GB/s	8.76 GB/s

Other than the three strategies, before FlashAttention [63], Megatron [8] proposed selective checkpointing that recomputes the core attention modules. As we use FlashAttention, the core attention module is done in one kernel, eliminating these intermediate tensors. The effect of selective checkpointing with FlashAttention has negligible impact on performance and peak memory usage for activations.

D. Discussion

Examining the modeling. To understand the accuracy of the model in Sec. III-D, we compare SSDTrain's offloaded amount with the model estimate. As Table III shows, the figures are close. We also compute the required PCIe write bandwidth, which is reduced when the hidden dimension gets larger. Typically, a model with more than 60B parameters has a hidden dimension of no less than 8K [35], [61]. PCIe write bandwidth aligns with the estimate in Sec. III-D.

Impact of larger micro-batch size. To further understand how larger micro-batch size improves the performance, we compare the no-offloading cases in Figure 7(a) to the same configurations with batch size as 1 and break down the throughput improvement in Fig. 8(a). The improvement primarily comes from time saving by weights update, which is very relevant to large-scale LLM training systems. The micro-batch size is usually set small, e.g., 1 or 2 in Paxml [64] and BLOOM [11] pretraining, in exchange for smaller bubbles introduced by PP. In the BLOOM training system, each data parallel rank is assigned a mini-batch with 32 samples. When the micro-batch size is no less than 4, the ideal PP bubble time percentage is no less than 11.5%. However, weight update and gradient accumulation cost is inversely proportional to the micro-batch size, which is huge when the micro-batch size is 1 or 2. SSDTrain allows larger micro-batch sizes given the same activation memory budget, thus beneficial to these PP-enabled training systems.

Impact of upscaling. Sec. II-B demonstrates that the whole-system activations size $S_{activations}$ grows slower than the whole-system GPU throughput C, i.e., $S_{activations} \propto C^{\frac{5}{6}}$. Therefore, the bandwidth required to fully overlap the computation with the SSD accesses is reduced. In short, the scaling of LLM is essentially a weak scaling scenario, and the SSD IO latency is easier to hide when it is scaled up. In Fig. 8(b), we further project the impact of upscaling on the write bandwidth per GPU using `llm-analysis`. We follow typical parallelism configurations [10], [64] when the number of GPUs is less than 100. In all projected cases, the write bandwidth per GPU is smaller than the original 2-GPU case (orange dashed line). Vanilla DP only affects weights update and therefore has no effect on the write bandwidth. ZeRO may reduce the write bandwidth requirement due to the communication incurred in forward and backward propagation.

Cost analysis. We study the SSD cost associated with adopting SSDTrain offloading. To get the endurance in Fig. 5, each A100 priced at US$10K [65] is paired with in total US$360 worth of SSDs. The evaluation uses 7 Intel P5800X for the 2 A100s. Although P5800X is more expensive, the price per PBW is comparable [66].

V. RELATED WORK

Many LLM systems with offloading abilities are inference-only [46], [47], [67]. In inference, weights and KV-cache never change and are reused across iterations; this is leveraged to enhance locality and memory efficiency. However, in training, all tensors, including the weights, change across the iterations. Some work avails offloading [48] for training but is mostly designed to fit in larger models at the cost of performance. Async transfer is missing to keep performance. Another direction is to offload computation to the CPU [22]–[24]. The offloaded computation is light, and the offloaded data include gradients, sparse elements in the weights, etc. Our work is orthogonal: Interference with the CPU is minimized because we offload the activations to SSDs via GDS. Activations are for compute-intensive gradient computation, which is best done solely on GPUs.

Before LLMs, there is work on offloading for deep learning [25], [68]–[71]: Most target main memory while some [25] target SSDs. LLM is unique because massive parallelism and its memory implications are fundamental to the design space. SSDTrain naturally supports multiple GPUs. We showed its viability on clusters. Besides, LLM's demand for computing power is so high that it stimulates rapid development in specialized hardware and frameworks. SSDTrain ensures good interoperability, while most prior work is bound to a specific PyTorch version or a custom runtime supporting select layers.

VI. CONCLUSION

In LLM training, activations dominate the increasingly limited GPU memory. To address this, we propose SSDTrain as an adaptive activation offloading framework to SSDs. We demonstrate its viability in large-scale systems by modeling. The evaluation shows SSDTrain reduces the activations peak memory use by up to 47% with negligible overhead. We analyze how this may lead to increased throughput by increasing micro-batch size and reducing pipeline bubbles.

ACKNOWLEDGEMENTS

This work was partially supported by the IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR), by the IBM-ILLINOIS Discovery Accelerator Institute (IIDAI), and by NVIDIA. This work was also partly supported by credits from Google Cloud, the Illinois Computes program, and the National Energy Research Scientific Computing Center (NERSC ERCAP0031237).

References

[1] OpenAI. (2022) ChatGPT. [Online]. Available: https://chatgpt.com/

[2] Microsoft. (2023) Bing Chat — Microsoft Edge. [Online]. Available: https://www.microsoft.com/en-us/edge/features/bing-chat

[3] Midjourney. (2022) Midjourney. [Online]. Available: https://www.midjourney.com/website

[4] LangChain. (2022) LangChain. [Online]. Available: https://github.com/langchain-ai/langchain

[5] J. Wei *et al.* (2022) Emergent Abilities of Large Language Models. [Online]. Available: http://arxiv.org/abs/2206.07682

[6] D. Meyer. (2024) The cost of training AI could soon become too much to bear. Yahoo Finance. [Online]. Available: https://finance.yahoo.com/news/cost-training-ai-could-soon-101348308.html

[7] Z. Liu *et al.*, "Winner-Take-All Column Row Sampling for Memory Efficient Adaptation of Language Model," Dec. 2023.

[8] V. Korthikanti *et al.*, "Reducing Activation Recomputation in Large Transformer Models," May 2022.

[9] Z. Jiang *et al.*, "MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs," Feb. 2024.

[10] M. Shoeybi *et al.*, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," Mar. 2020.

[11] T. L. Scao *et al.*, "BLOOM: A 176B-Parameter Open-Access Multilingual Language Model," Jun. 2023.

[12] L. Chen, "Dissecting Batching Effects in GPT Inference," https://le.qun.ch/en/blog/2023/05/13/transformer-batching/, 2023.

[13] Q. Anthony *et al.*, "The Case for Co-Designing Model Architectures with Hardware," Jan. 2024.

[14] R. Y. Aminabadi *et al.*, "DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale," Jun. 2022.

[15] J. Kaplan *et al.*, "Scaling Laws for Neural Language Models," Jan. 2020.

[16] S. McCandlish *et al.*, "An Empirical Model of Large-Batch Training," Dec. 2018.

[17] T. Chen *et al.*, "Training Deep Nets with Sublinear Memory Cost," Apr. 2016.

[18] The Epoch AI, "Announcing Epoch AI's Updated Parameter, Compute and Data Trends Database," Oct. 2023.

[19] Microsoft, "ND A100 v4-series - Azure Virtual Machines," Feb. 2024.

[20] Google, "GPU machine types | Compute Engine Documentation," 2017.

[21] NCSA, "Delta Project Profile," https://www.ncsa.illinois.edu/research/project-highlights/delta/, 2022, accessed 07/21/2024.

[22] K. Kamahori *et al.* (2024) Fiddler: CPU-GPU Orchestration for Fast Inference of Mixture-of-Experts Models.

[23] J. Ren *et al.*, "ZeRO-Offload: Democratizing Billion-Scale Model Training," in *USENIX ATC*, 2021.

[24] Y. Song *et al.* (2023) PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. [Online]. Available: http://arxiv.org/abs/2312.12456

[25] J. Bae *et al.*, "FlashNeuron: SSD-Enabled Large-Batch Training of Very Deep Neural Networks," in *FAST*, 2021.

[26] Google, "About google cloud hyperdisk — compute engine documentation," https://cloud.google.com/compute/docs/disks/hyperdisks, 2022.

[27] G. K. Lockwood *et al.*, "Architecture and performance of Perlmutter's 35 PB ClusterStor E1000 all-flash file system," *CCPE*, p. e8143, 2024.

[28] Microsoft. (2019) Megatron-DeepSpeed: Ongoing research training transformer language models at scale, including: BERT & GPT-2.

[29] A. Radford *et al.* (2019) Language Models are Unsupervised Multitask Learners.

[30] A. Vaswani *et al.*, "Attention is All you Need," in *NeurIPS*, vol. 30. Curran Associates, Inc., 2017.

[31] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," May 2019.

[32] C. Raffel *et al.*, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," Sep. 2023.

[33] Y. Xu *et al.*, "GSPMD: General and Scalable Parallelization for ML Computation Graphs," Dec. 2021.

[34] S. Rajbhandari *et al.*, "ZeRO: Memory optimizations Toward Training Trillion Parameter Models," in *SC20*. IEEE, Nov. 2020, pp. 1–16.

[35] Jordan Hoffmann *et al.* (2022) Training Compute-Optimal Large Language Models. [Online]. Available: http://arxiv.org/abs/2203.15556

[36] T. B. Brown *et al.*, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[37] Samsung, "Ultra-Low Latency with Samsung Z-NAND SSD," https://download.semiconductor.samsung.com/resources/brochure/Ultra-LowLatencywithSamsungZ-NANDSSD.pdf, 2017.

[38] JEDEC, *JESD218B: Solid-State Drive (SSD) Requirements and Endurance Test Method*, Std., 2016.

[39] Lenovo. (2023) What do I need to know about SSD endurance and overprovisioning? [Online]. Available: https://thinksystem.lenovofiles.com/storage/help/index.jsp?topic=%2Fde-series-olh-11.80%2Fwhat-do-i-need-to-know-about-ssd-endurance-and-overprovisioning.html

[40] QNAP Systems, Inc., "QNAP NAS Solution: QTS SSD Extra Over-Provisioning," 2018.

[41] SMART Modular Technologies, Inc. (2024) Why SMART's Over-Provisioning? [Online]. Available: https://www.smartm.com/technology/over-provisioning

[42] Solidigm. (2022) Solidigm™ SSD Endurance Estimator. [Online]. Available: https://estimator.solidigm.com/ssdendurance/index.htm

[43] Intel. (2018) Over-Provisioning NAND-Based Intel® SSDs for Better Endurance. [Online]. Available: https://www.ioncomputer.com/ion/body/documents/over-provisioning-nand-based-ssds-better-endurance-whitepaper.pdf

[44] Samsung, "Over-Provisioning Benefits for Samsung Data Center SSDs," 2019. [Online]. Available: https://download.semiconductor.samsung.com/resources/white-paper/S190311-SAMSUNG-Memory-Over-Provisioning-White-paper.pdf

[45] S. Maneas *et al.*, "Operational Characteristics of SSDs in Enterprise Storage Systems: A Large-Scale Field Study," in *FAST*, 2022.

[46] Y. Sheng *et al.*, "FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU," Jun. 2023.

[47] K. Alizadeh *et al.*, "LLM in a flash: Efficient Large Language Model Inference with Limited Memory," Jan. 2024.

[48] S. Rajbhandari *et al.*, "ZeRO-infinity: Breaking the GPU memory wall for extreme scale deep learning," in *SC*, Nov. 2021, pp. 1–14.

[49] D. Inupakutika *et al.*, "Quantifying Performance Gains of GPUDirect Storage," in *NAS*, 2022, pp. 1–9.

[50] H. Yang *et al.* (2024) ProTrain: Efficient LLM Training via Memory-Aware Techniques. [Online]. Available: http://arxiv.org/abs/2406.08334

[51] X. Sun *et al.*, "STRONGHOLD: Fast and affordable billion-scale deep learning model training," in *SC*, 2022, pp. 1–17.

[52] Nvidia, "Rapidsai/kvikio: KvikIO - High Performance File IO," https://github.com/rapidsai/kvikio, 2022, accessed 07/21/2024.

[53] Wikipedia, "Monkey patch," 2006.

[54] C. Li, "LLM-Analysis: Latency and Memory Analysis of Transformer Models for Training and Inference," 2023.

[55] Y. Cai *et al.*, "Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime," in *ICCD*, 2012.

[56] ——, "Error patterns in mlc nand flash memory: Measurement, characterization, and analysis," in *DATE*. IEEE, 2012, pp. 521–526.

[57] R.-S. Liu *et al.*, "Optimizing NAND Flash-Based SSDs via Retention Relaxation," in *FAST*. USENIX Association, 2012.

[58] S. Kim *et al.*, "Behemoth: A Flash-centric Training Accelerator for Extreme-scale dnns," 2021, pp. 371–385.

[59] P. J. O. Su'arez *et al.*, "A monolingual approach to contextualized word embeddings for mid-resource languages," in *ACL*, Jul. 2020.

[60] ——, "Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures," ser. CMLC-7. Mannheim: Leibniz-Institut f'ur Deutsche Sprache, 2019, pp. 9 – 16.

[61] H. Touvron *et al.*, "Llama 2: Open Foundation and Fine-Tuned Chat Models," Jul. 2023.

[62] T. Dao. (2023) FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning.

[63] T. Dao *et al.* (2022) FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness.

[64] Nitin *et al.* (2023) Scaling large language model training with Pax on GPUs.

[65] Dihuni. (2021) NVIDIA A100 40GB Ampere PCIe GPU.

[66] Newegg. (2021) Intel optane p5800x 1.6TB.

[67] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *SOSP*. ACM, 2023.

[68] X. Peng *et al.*, "Capuchin: Tensor-based GPU Memory Management for Deep Learning," in *ASPLOS*. ACM, Mar. 2020, pp. 891–905.

[69] L. Wang *et al.*, "SuperNeurons: Dynamic GPU Memory Management for Training Deep Neural Networks," in *PPOPP*, Feb. 2018, pp. 41–53.

[70] M. Rhu *et al.*, "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design," Jul. 2016.

[71] C.-C. Huang *et al.*, "SwapAdvisor: Pushing Deep Learning Beyond the GPU Memory Limit via Smart Swapping," in *ASPLOS*, Mar. 2020.

3D-CIMlet: A Chiplet Co-Design Framework for Heterogeneous In-Memory Acceleration of Edge LLM Inference and Continual Learning

Shuting Du[1], Luqi Zheng[1], Aradhana Mohan Parvathy[1], Feifan Xie[2], Tiwei Wei[2], Anand Raghunathan[1], Haitong Li[1]
[1]Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA
[2]School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907, USA
{du335, zheng782, amohanpa, xie477, wei427, araghu, haitongli}@purdue.edu

***Abstract*—The design space for edge AI hardware supporting large language model (LLM) inference and continual learning is underexplored. We present 3D-CIMlet, a thermal-aware modeling and co-design framework for 2.5D/3D edge-LLM engines exploiting heterogeneous computing-in-memory (CIM) chiplets, adaptable for both inference and continual learning. We develop memory-reliability-aware chiplet mapping strategies for a case study of edge LLM system integrating RRAM, capacitor-less eDRAM, and hybrid chiplets in mixed technology nodes. Compared to 2D baselines, 2.5D/3D designs improve energy efficiency by up to 9.3x and 12x, with up to 90.2% and 92.5% energy-delay product (EDP) reduction respectively, on edge LLM continual learning.**

***Index Terms*—Transformers, Memory-Centric Computing, Continual Learning, Heterogeneous Integration, Chiplets**

I. Introduction

The rapid growth in large language models (LLMs) has fueled a pressing need for efficient on-device inference and learning solutions. The structural differences between LLM training, fine-tuning, inference, and continual learning introduce unique design challenges that have been overlooked in prior efforts. These challenges, particularly in dataflow, memory access, and computation, represent key gaps in the development of hardware accelerators for on-device training [1]. As depicted in Fig.1, BERT and Adapter-BERT [2] exhibit structural differences that highlight distinct hardware requirements for continual learning and fine-tuning. To meet the demands of high performance, flexibility, and adaptability, a large amount of edge ML accelerator studies focused on co-design integrating architectural and technological innovations. These include circuit designs leveraging memory-centric architectures based on silicon [3]–[5] or non-volatile memory (NVM) [6], [7], as well as advanced modeling and design exploration tools [8]–[11]. Furthermore, scaling and adapting the single-chip designs to multi-die architectures [11]–[13] has proven effective in bridging the gap between resource-constrained edge and high-performance edge systems capable of LLM workloads. However, despite rapid advancements across the technology-to-system stack, significant challenges persist in achieving efficient on-device LLM inference and continual learning, largely due to the absence of comprehensive co-design frameworks.

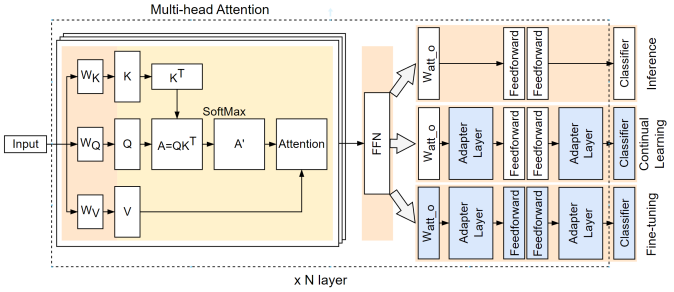


Fig. 1. Transformer-based LLMs create diverse requirements for edge inference and learning, adapter-based continual learning, fine-tuning, and inference layer structures highlighted in our case study.

In this work, a 2.5D/3D thermal-aware co-design framework leveraging heterogeneous in-memory computing is developed to design efficient chiplet-based edge LLM engines supporting inference and continual learning. With the developed co-design and modeling framework, efficient edge LLM inference and continual learning are demonstrated through a case study of reliability-aware heterogeneous eDRAM/RRAM chiplet designs in cost-effective, mixed-technology nodes. The framework is open-sourced.[1] The contribution of this work is summarized as follows:

- We develop 3D-CIMlet, a modeling and co-design framework that allows rapid design space exploration of 2.5D/3D chiplet-based accelerator architectures for transformers, leveraging heterogeneous memory technologies.
- Based upon 3D-CIMlet, we develop a heterogeneous RRAM/eDRAM CIM system with 2.5D and 3D integration schemes and corresponding reliability-aware mapping strategies to support efficient inference and continual learning of edge LLMs.
- Through chiplet-to-package, multi-scale design space explorations (DSE), we provide co-optimization guidelines spanning CIM chiplet designs (intra-chiplet and inter-chiplet), cost-aware and thermal-aware system integration, and runtime optimizations for continual learning.

[1]https://github.com/NanoX-Lab/3D-CIMlet

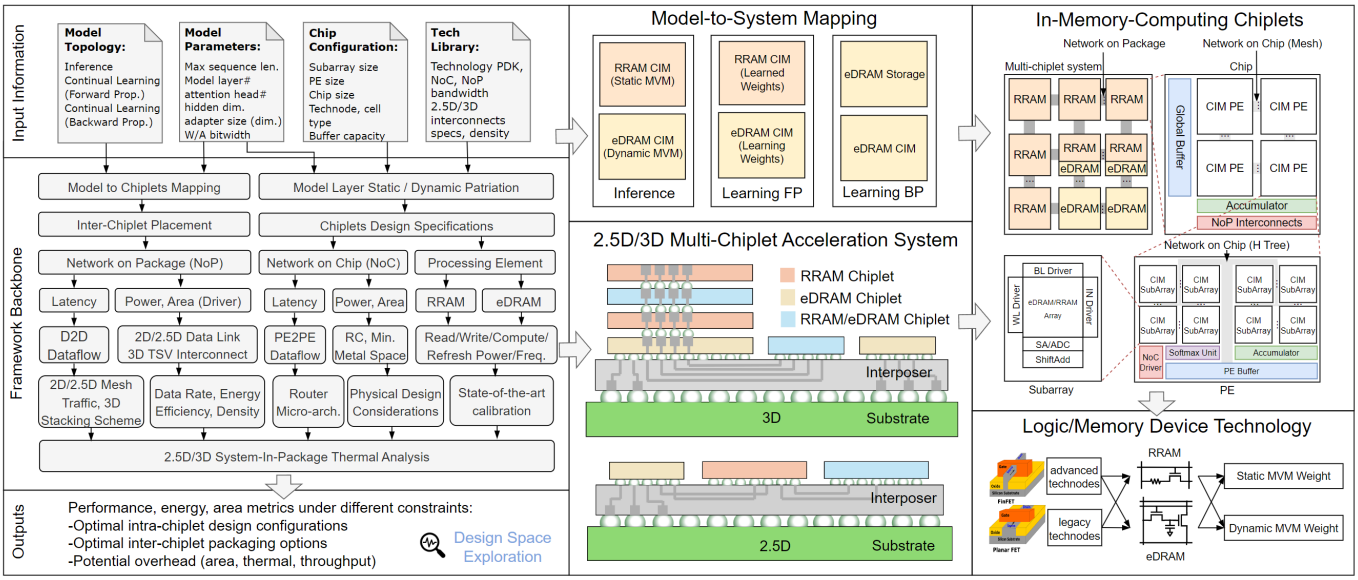


Fig. 2. Overview of 3D-CIMlet, a 2.5D/3D chiplet-based modeling and co-design framework for edge LLM inference and continual learning.

II. Background and Related Works

Single-chip accelerators for transformer-based models, such as SRAM-based CIM accelerators [5] and RRAM-based accelerators capable of storing all model weights [14], have shown promise for LLM edge applications. However, these single-chip solutions face limitations: they can only parallelize a few transformer layers [5] and accommodate very tiny models [14] within the on-chip embedded NVM. Benefiting from the advanced packaging technologies, multi-chiplet systems combine the strengths of mature single-die transformer accelerator designs with high-density edge NVM storage, enabling more robust and scalable edge LLM applications.

Multi-chiplet architectures have improved the performance of deep neural networks (DNNs) inference and learning significantly in the past decades. CIM architecture based prototypes include the CHIMERA Illusion System [12] and the multi-chip module (MCM) TensorCIM [13]. Moreover, [11] proposed a 2.5D chiplet-based RRAM-CIM accelerator simulator and benchmarking tool for DNNs. Despite these advancements, current architectures that rely on single-chiplet design foundations have not been fully expanded to support diverse chiplet configurations across systems to optimize for LLM workloads.

Previous modeling frameworks have primarily focused on single-die accelerators, leaving multi-die designs for edge LLM applications largely unexplored. Tools such as Timeloop/Accelergy [8] and AccelTran [9] targeted DNNs and sparsity-aware inference, respectively, but are limited to single-die architectures and inference scenarios. 3D Neurosim [10] explored advanced packaging with CIM architectures but remains limited in scalability for complex chiplet systems. Similarly, SIAM [11] benchmarked 2.5D RRAM-CIM accelerators but lacks support for continual learning workloads which are essential for edge LLMs.

III. 3D-CIMlet: Co-Design Framework

A. Methodology

3D-CIMlet is a two-prong co-design framework: (1) Exploiting transformer CIM designs with various chiplet architectures through advanced packaging, multi-scale technology-to-system design space is fully exposed. Thermal-aware design space explorations (DSE) can be conducted to probe into design tradeoffs and optimization opportunities. (2) Building upon the distributed memory-centric architectures, the framework explores memory-reliability-aware mapping strategies for both inference and continual learning. This is achieved by connecting inference and continual learning characteristics of edge LLMs to the heterogeneous memory characteristics from the chiplet primitives.

Figure 2 provides an overview of 3D-CIMlet methodology and key backbones. The framework takes transformer model inputs and calibrated technology libraries. The technology libraries include silicon and beyond-silicon computational memories, die-to-die (D2D) interconnects, and 2.5D/3D integration technologies as the foundational enablers. The scope of this work is focused on a case study analyzing heterogeneous computational memories in heterogeneous architectures. Hence, we architect the edge LLM system consisting of resistive RAM (RRAM), capacitor-less embedded DRAM (eDRAM), and hybrid RRAM/eDRAM chiplets designed as modular CIM primitives at both chiplet level and system level. The network-on-package (NoP) and network-on-chip (NoC) hierarchies support D2D interconnection and routing of CIM processing engines (PEs). Each CIM chiplet contains CIM PEs for matrix-vector multiplication (MVM), global buffers, accumulators, and NoP drivers. Each CIM PE includes CIM subarrays exploiting charge-based or non-charge-based computations, softmax units, accumulators, buffers and NoC drivers. Since

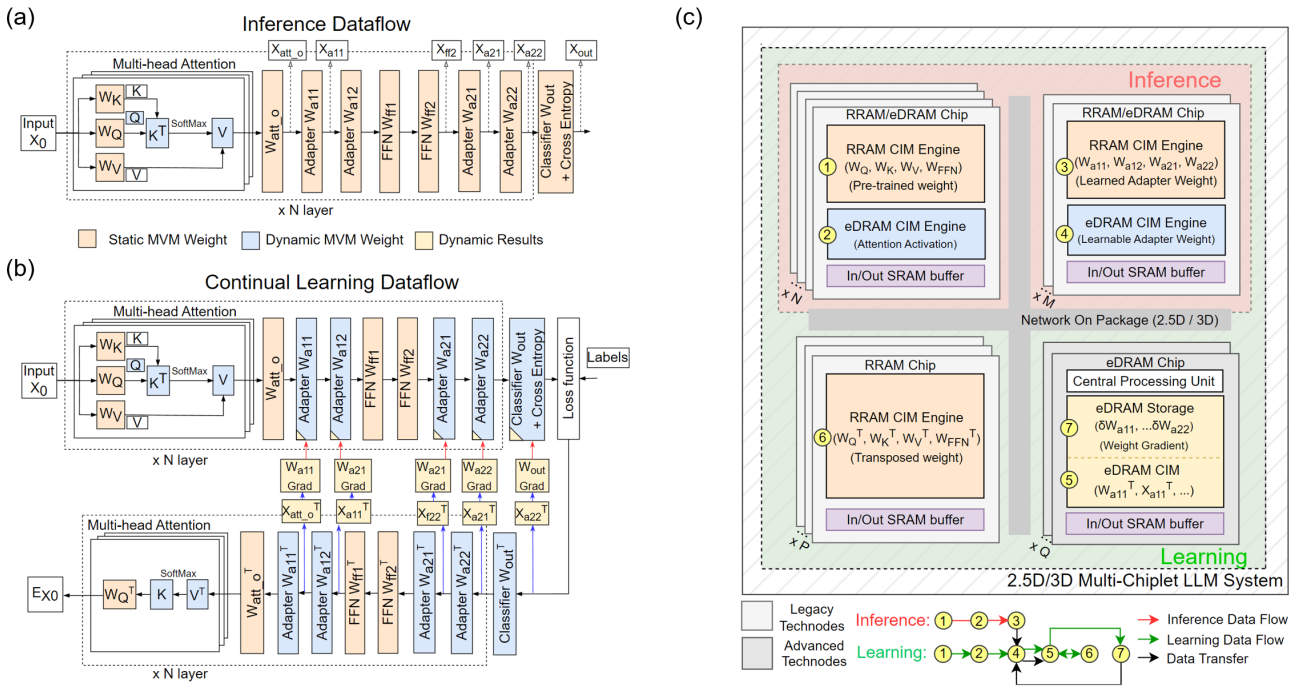


Fig. 3. The dataflow and static/dynamic operation partitioning in adapter-based (a) inference and (b) continual learning, together with (c) a reliability-aware mapping strategy for the heterogeneous multi-chiplet edge LLM system. With mixed technology nodes supported, RRAM/eDRAM chiplets are leveraged for inference and forward propagation during learning, while eDRAM chiplets are utilized for backpropagation.

this work focuses on edge LLM use cases, our case study takes a relatively conservative, cost-driven assumption to land on a sweet spot between cost [15], [16], embodied carbon footprint [17], and system efficiency: RRAMs are based on 40 nm foundry RRAM technology in this cost-effective legacy node [6], [12], [18], whereas capacitor-less eDRAMs being used are mature in both planar [3], [4] and FinFET (14/16 nm) nodes [19], [20]. For intra-chiplet NoC and inter-chiplet NoP modeling in the backbone of the framework, BookSim [21] is customized incorporating the 2D [22], 2.5D [23], [24], and 3D [25]–[27] system integration technologies.

First, for intra-chiplet design explorations, the framework categorizes and compiles input model layers into static and dynamic operations, and optimizations around those operations are guided by heterogeneous memory characteristics seen in silicon charge-based memories as well as beyond-silicon non-volatile memories (NVMs) across chiplets. In our case study, RRAM and eDRAM technologies form a wide "reliability spectrum": capacitor-less eDRAMs with unlimited endurance would suffer from limited retention leading to frequent refreshes, whereas excessive writes to RRAMs with long retention must be minimized. NoC behaviors and performance are simulated in a cycle-accurate fashion, focusing on PE-to-PE dataflow and router architecture, with power and area estimations based on RC considerations and minimum metal spacing. Second, inter-chiplet design exploration is associated with model-to-chiplet mapping strategies (Section IV). NoP behaviors and performance are analyzed considering D2D dataflow and traffic patterns, including 2D/2.5D mesh and 3D stacking schemes. Power and area are further modeled based on 2D/2.5D data links and 3D TSV interconnects [23]–[27].

B. Chiplet-to-Package Thermal Modeling

Finite Element Method (FEM) is employed to thoroughly analyze the thermal characteristics and performance of both 2.5D and 3D integration. The front end of line (FEOL) is modeled as a surface heat source, with all chiplets mounted on a 100 µm thick silicon interposer substrate. The chips are mechanically reinforced and encapsulated with an adiabatic molding compound. Heat transfer coefficients of 1000 W/(m²·K) on the top of silicon dies and 20 W/(m²·K) on the bottom of the organic substrate are used, reflecting moderate forced convection cooling on the package surface.

IV. Reliability-Aware Chiplet Mapping Strategies

The 3D-CIMlet framework aims to enable efficient chiplet design and system integration by combining inference and continual learning acceleration within a single chiplet-based system. Hardware resources are allocated dynamically to inference and continual learning tasks as needed. The 40 nm RRAM, 40 nm RRAM/eDRAM, and 14 nm eDRAM chiplets serve as modular, reusable components that facilitate the mapping of MVM operations and on-chip data storage within an integrated edge LLM engine (Fig. 3). Additionally, for inference-only edge use cases, the chiplet-based approach allows for a cost-effective, scaled-back version with chiplets in low-cost technology node only.

The dataflow of inference with learnable adapter layers is shown in Fig. 3(a). Adapter layers are lightweight modules

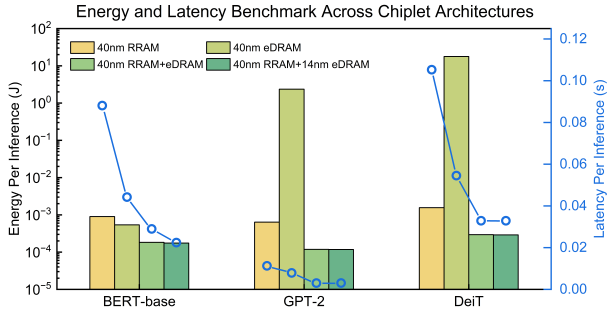


Fig. 4. The energy consumption and latency benchmark across different chiplet architectures for inference workloads. Heterogeneous RRAM/eDRAM designs improve the total inference energy by up to 1.4x and the inference latency by up to 3.9x from RRAM-CIM chiplet baseline.

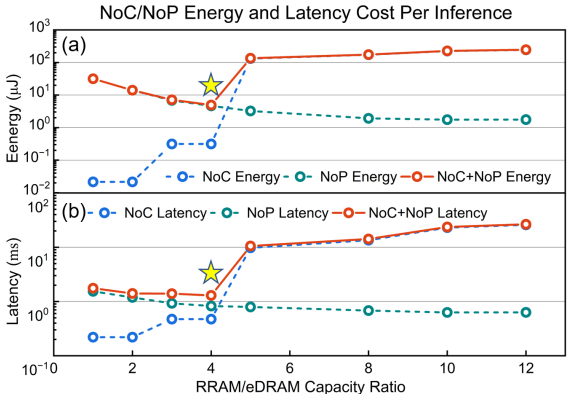


Fig. 5. The NoC and NoP energy consumption and latency associated with 40 nm RRAM/eDRAM chiplets with varying RRAM-to-eDRAM capacity ratios. (a) NoC and NoP energy cost per inference, (b) NoC and NoP latency per inference. At a ratio of 4, the balance of NoC and NoP is reached with minimized communication overheads.

added into pre-trained backbone networks, adapting the models to new tasks without modifying the original pre-trained weights. During inference, apart from the dynamic MVM operations between K, Q, and V in the attention layers of transformers, all other weights remain static since they belong to the pre-trained backbone and adapter layers. As shown in Fig. 3(c), inference workloads are mapped onto 40 nm RRAM/eDRAM chiplets with the wide retention spectrum: static MVM weights are handled by RRAM CIM engines, including the weights of K, Q, V projection layers, feedforward layers ①, and the trained adapter layers ③, which remain the same with varying input activations. The dynamic attention activations ② varying with input activations are allocated to eDRAM CIM engines without endurance limitations while incurring the trade-off of short retention times.

In addition to the RRAM/eDRAM chiplets accelerating inference, 14 nm capacitor-less eDRAM and 40 nm RRAM chiplets are leveraged to expand the system for continual learning, with the dataflow and mapping shown in Fig. 3(b) and Fig. 3(c). Forward propagation (FP) during learning is executed on the same chiplets allocated for inference, with the key distinction that adapter weights are learnable during FP as the signature of continual learning. To avoid endurance issues with RRAM during weight updates in backpropagation (BP), these learnable weights are initially loaded from the RRAM CIM engines ③ to the eDRAM CIM engines ④. BP operations are distributed across RRAM chiplets and 14 nm eDRAM chiplets. Errors propagated backward are multiplied by the transposed adapter weights and the transposed FP output activations to compute weight gradients. These computations occur in the endurance-optimal eDRAM CIM engines ⑤, with transposed static weights in the retention-optimal RRAM CIM engines ⑥. In contrast to high-bandwidth memory (HBM) based designs, the 14 nm eDRAM chiplets allow on-die integration with a processor unit, which orchestrates the process of storing the generated weight gradients in eDRAM chiplets ⑦. These gradients are retained until needed for weight updates during the subsequent FP iteration in the eDRAM CIM engines ④.

V. Design Explorations and Analysis

A. Edge LLM Inference

For edge LLM inference, we evaluated models across two key aspects: (1) different input modalities (text and image) which require distinct embedding strategies, and (2) diverse transformer architectures, including encoder-only and decoder-only models. Language and vision models differ in embedding approaches; language models use embedding tables, whereas vision models often rely on convolutional layers. Our experiments included a convolution-based embedding layer for vision transformers to reflect this difference. Due to their autoregressive nature, decoder-only models process one query at a time and perform vector-matrix computations, in contrast to encoder-only models, which predominantly have matrix-matrix multiplications. Based on these distinctions, we evaluated a diverse suite of models: (1) BERT-base, an encoder-only text model, on the GLUE dataset (average of 128 tokens); (2) GPT-2, a decoder-only text generation model (128 tokens and 1024 tokens) on the Wikitext-2 dataset; (3) DeiT-base, an encoder-only image model, on ImageNet (196 tokens).

We illustrate 3D-CIMlet for inference through two sets of DSE analyses. First, we analyze energy consumption and performance across various chiplet configurations for inference workloads, including BERT, GPT-2, and DeiT. As shown in Fig. 4, across transformer model architectures, heterogeneous chiplet designs with RRAM and eDRAM available in the system demonstrate up to 3.9x and 2.6x performance and up to 1.4x and more than 10^4x energy efficiency improvement compared to those with NVMs or charge-based memories only.

Second, RRAM-to-eDRAM capacity ratio in 40 nm RRAM/eDRAM chiplets is a key design knob in our DSE experiments. To better understand the overall impact of NoP and NoC, their energy and latency overheads are evaluated. As shown in Fig. 5, the NoC energy and latency costs per inference gradually increase, while the NoP energy and latency decrease as the RRAM-to-eDRAM capacity ratio scales. This analysis reveals that an optimal balance in communication

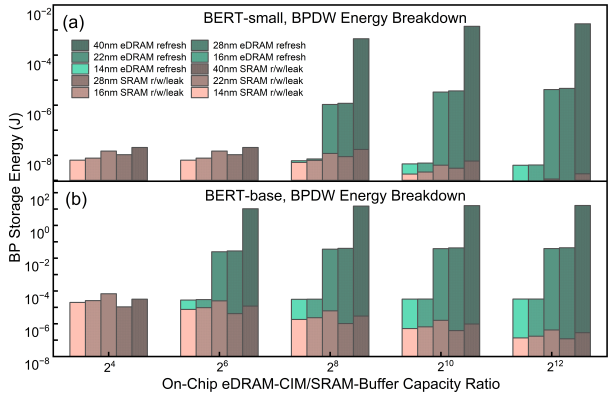


Fig. 6. Energy costs of backpropagation dynamic weight (BPDW) storage across various eDRAM-to-SRAM capacity ratios on eDRAM chiplets, in multiple technology nodes, analyzed for continual learning in (a) BERT-small and (b) BERT-base models. SRAM leakage and eDRAM refresh play key roles in BPDW memory optimization for continual learning workloads of different scales.

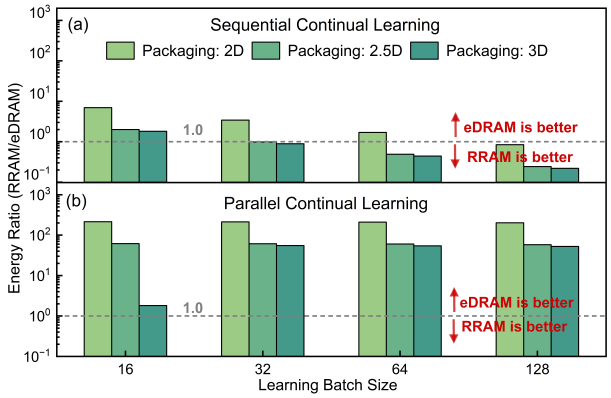


Fig. 7. Relative energy costs for weight gradient storage between 40 nm RRAM (due to write-back) and 14 nm eDRAM (due to refresh) in (a) sequential-mode continual learning and (b) parallel-mode continual learning, analyzed across various integration schemes.

overhead between NoC and NoP can be obtained at a ratio of 4 for RRAM/eDRAM capacity allocation across chiplets.

B. Edge LLM Continual Learning

For continual learning with 3D-CIMlet, Adapter-BERT is evaluated on two configurations: (1) BERT-base (12 layers, 12 attention heads) on the GLUE dataset (average 128 tokens), and (2) BERT-small (4 layers, 4 attention heads, average 32 tokens). During continual learning tasks, layer output activations during FP act as dynamic weights during BP, referred to as backpropagation dynamic weights (BPDW), alongside learnable adapter weights. The required data retention time for BPDW, in the context of eDRAM refreshes, varies by layer position within or across encoder/decoder modules. Similarly, the retention time for weight gradients before the next FP iteration depends on the learning mode (sequential or parallel) and batch size. Sequential learning mode processes input samples without overlapping layer computations within a single iteration. This approach trades off longer retention times for weight gradients with potentially lower system power. In contrast, parallel learning mode pipelines input samples across multiple layers simultaneously. 3D-CIMlet orchestrates on-chip BP data storage through DSE on two fronts: (1) exploring and optimizing the on-die eDRAM-to-SRAM buffer capacity ratio in eDRAM chiplets of multiple technology nodes for BPDW storage, and (2) optimizing the on-chip storage of weight gradients between RRAM and eDRAM chiplets.

For continual learning, each eDRAM chiplet is configured with storage and CIM partitions to store and process BPDW. With on-die SRAM buffers, we first evaluate the impact of hybrid SRAM/eDRAM storage. A mapping strategy is taken such that early-generated BPDW are stored in an SRAM buffer until full, with subsequent weights stored directly in eDRAM on the same die. SRAM-stored weights are transferred to the eDRAM-CIM for computation as needed, while eDRAM-stored weights are refreshed until the next computation. For the BERT-small model with shorter iteration durations, reduced energy costs are observed at 14 nm and 16 nm nodes with decreasing SRAM buffer capacities (Fig.6(a)). In contrast, for the BERT-base model, increasing the eDRAM/SRAM capacity ratio leads to higher energy costs for BPDW storage across planar and FinFET silicon nodes (from 40 nm to 14 nm), due to low SRAM leakage offset by high eDRAM refresh energy over time (Fig.6(b)). These results suggest that advanced-node eDRAM, with low refresh power, is better suited for BPDW storage in small-scale continual learning scenarios. For larger-scale continual learning, increasing SRAM buffer sizes reduces energy costs despite SRAM leakage, while the density benefit of capacitor-less eDRAMs mitigates the area overhead of larger SRAM buffers within a given BPDW energy budget.

Second, the weight gradient storage options under sequential and parallel learning modes are analyzed to identify runtime optimization opportunities for continual learning workloads (Fig. 7). In sequential mode (less thermal stress), the required retention time for weight gradients decreases with smaller learning batch sizes, which increases the likelihood of using eDRAM. For larger batch sizes, high refresh energy costs of eDRAM tend to push the storage needs towards RRAM, especially with low energy overhead of 2.5D/3D packaging. In parallel mode (higher performance), the inherently relaxed retention requirement results in eDRAM being the preferred storage medium across all batch sizes and integration schemes.

In broader use cases, continual learning and inference tasks are periodically interleaved. 3D-CIMlet's DSE provides insights into the impact of different chiplet stacking architectures on energy efficiency and energy-delay product (EDP) across various model architectures and application scenarios (Fig. 8).

For inference (Fig. 8(a)), 2.5D/3D architectures significantly improve energy efficiency and latency of traditional 2D packages, with TOPS/W increased by 9.9× for BERT-base and 4.5× for DeiT-base, compared to 1.1× for GPT-2. The primary driver of these energy efficiency and latency gains with 3D stacking is reduced communication overhead from enhanced D2D connectivity. The D2D communication energy in BERT-base and DeiT-base constitutes 90.8% and 78.3% of total

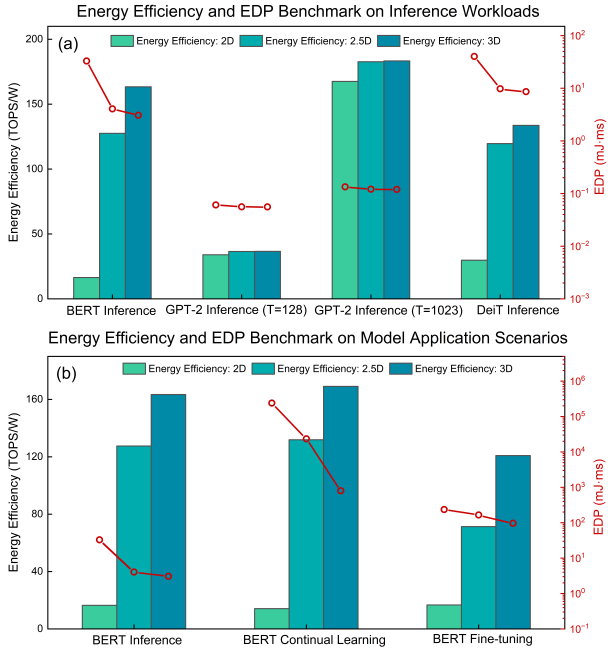


Fig. 8. The energy efficiency and energy-delay product (EDP) of 3D-CIMlet designs for (a) inference with 40 nm RRAM/eDRAM chiplets, and (b) Adapter-BERT inference, continual learning, and fine-tuning with integrated 14 nm eDRAM and 40 nm RRAM/eDRAM chiplets.

energy in the 2D architecture, significantly benefiting from chiplet integration. In GPT-2 architecture, the autoregressive design leads to sequential token generation, resulting in negligible D2D communication costs compared to computation. NoP accounts for only 8.7% of total energy during 1024-token inference with modest TOPS/W improvements.

For continual learning (Fig. 8(b)), 9.3× and 12.0× improvements in energy efficiency, along with a 90.2% and 92.5% reduction in EDP, respectively, are achieved by the 2.5D and 3D designs through the 3D-CIMlet framework compared to 2D baselines. The inference exhibits fewer gains with only 40 nm RRAM/eDRAM chiplets. For fine-tuning, the rise in on-chip computations outweighs the increase in NoP communication costs, resulting in diminished improvements overall.

C. Thermal Analysis

Chiplet-to-package thermal analysis plays a pivotal role in edge LLM continual learning by highlighting key thermal challenges and trade-offs in managing temperatures for advanced 2.5D/3D designs. Figure 9 illustrates the packaging-level thermal modeling setup (a), and the peak temperature increase contours with various packaging schemes (b)–(d). A non-uniform temperature distribution is observed due to varying power characteristics across chiplets. Thermal modeling reveals distinct temperature profiles for 2.5D/3D packaging schemes. In the 2.5D setup, the maximum temperature increase ($T_{\max}$) reaches 29.3 K, with a temperature difference (ΔT) of 12.2 K, indicating moderate thermal stress. However, persistent hotspots are found in the 14 nm eDRAM chiplet due to high power density and limited cooling pathways.

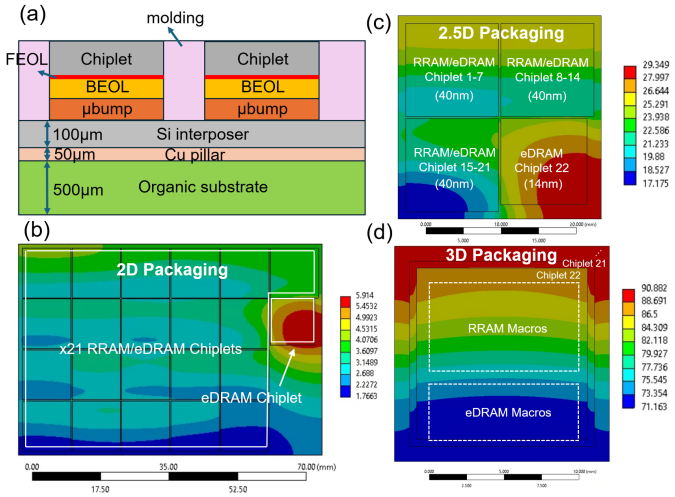


Fig. 9. Chiplet-to-package thermal analysis for edge LLM continual learning across various packaging configurations. (a) Schematic of the 2.5D/3D packages for thermal modeling and analysis. Peak temperature increase contours are generated across three different packaging designs: (b) traditional 2D packaging layout with a large form factor, (c) 2.5D chiplet integration with limited die stacking, and (d) full 3D stacking.

Adopting a worst-case scenario, full 3D stacking shows highest thermal stress driven by increased thermal resistance and inefficient heat dissipation, calling for additional cooling to mitigate thermal buildup. It is worth noting that the on-die memory area allocation between RRAM and eDRAM macros in the 3D chiplets and the package design jointly lead to non-uniform peak temperature distribution with cooler eDRAMs, which helps mitigate the impact on eDRAM retention. Comparatively, 2.5D designs offer better temperature uniformity compared to 3D designs but still face challenges with localized thermal hotspots, trading off system-level compute density.

VI. Conclusion

We present 3D-CIMlet, a co-design framework that harnesses the unique capabilities of heterogeneous CIM chiplets within 2.5D/3D multi-die architectures, enabling efficient inference and continual learning for edge LLMs. 2.5D and 3D designs with heterogeneous RRAM/eDRAM chiplets lead to significant energy efficiency and EDP benefits compared to 2D architecture for continual learning. At the core of 3D-CIMlet's co-design capabilities are diverse embedded computational memories, in-memory compute-storage allocation strategies, NoP/NoC interplays with intra-chiplet designs, flexible model-to-architecture mapping space, and chiplet-to-package thermal analysis. These features, extending beyond the case study presented, will enable memory-reliability-aware and thermal-aware system designs for scalable and energy-efficient deployment of future LLM workloads at the edge and beyond.

Acknowledgment

This work was supported in part by the U.S. National Science Foundation under Award No. 2425498 with industry partners as specified in the Future of Semiconductors (FuSe2)

program, in part by the UPWARDS for the Future Network program (with funding from NSF Award No. 2329784, Micron Technology, and TEL), and also in part by CoCoSys, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) Program sponsored by DARPA.

REFERENCES

[1] J. Lee and H.-J. Yoo, "An overview of energy-efficient hardware accelerators for on-device deep-neural-network training," *IEEE Open Journal of the Solid-State Circuits Society*, vol. 1, pp. 115–128, 2021.

[2] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. D. Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 2790–2799.

[3] Z. Chen, X. Chen, and J. Gu, "15.3 a 65nm 3t dynamic analog ram-based computing-in-memory macro and cnn accelerator with retention enhancement, adaptive analog sparsity and 44tops/w system energy efficiency," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 240–242.

[4] Y. He, S. Fan, X. Li, L. Lei, W. Jia, C. Tang, Y. Li, Z. Huang, Z. Du, J. Yue, X. Li, H. Yang, H. Jia, and Y. Liu, "34.7 a 28nm 2.4mb/mm2 6.9 - 16.3tops/mm2 edram-lut-based digital-computing-in-memory macro with in-memory encoding and refreshing," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 578–580.

[5] F. Tu, Z. Wu, Y. Wang, L. Liang, L. Liu, Y. Ding, L. Liu, S. Wei, Y. Xie, and S. Yin, "Trancim: Full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 6, pp. 1798–1809, Jun. 2023.

[6] S. D. Spetalnick, A. S. Lele, B. Crafton, M. Chang, S. Ryu, J.-H. Yoon, Z. Hao, A. Ansari, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, and A. Raychowdhury, "30.1 a 40nm vliw edge accelerator with 5mb of 0.256pj/b rram and a localization solver for bristle robot surveillance," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 482–484.

[7] C.-X. Xue, Y.-C. Chiu, T.-W. Liu, T.-Y. Huang, J.-S. Liu, T.-W. Chang, H.-Y. Kao, J.-H. Wang, S.-Y. Wei, C.-Y. Lee *et al.*, "A cmos-integrated compute-in-memory macro based on resistive random-access memory for ai edge devices," *Nature Electronics*, vol. 4, no. 1, pp. 81–90, 2021.

[8] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2019.

[9] S. Tuli and N. K. Jha, "Acceltran: A sparsity-aware accelerator for dynamic inference with transformers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4038–4051, 2023.

[10] X. Peng, W. Chakraborty, A. Kaul, W. Shim, M. S. Bakir, S. Datta, and S. Yu, "Benchmarking monolithic 3d integration for compute-in-memory accelerators: overcoming adc bottlenecks and maintaining scalability to 7nm or beyond," in *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2020, pp. 30–4.

[11] G. Krishnan, S. K. Mandal, M. Pannala, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao, "Siam: Chiplet-based scalable in-memory acceleration with mesh for deep neural networks," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 5s, pp. 1–24, Oct. 2021.

[12] K. Prabhu, A. Gural, Z. F. Khan, R. M. Radway, M. Giordano, K. Koul, R. Doshi, J. W. Kustin, T. Liu, G. B. Lopes, V. Turbiner, W.-S. Khwa, Y.-D. Chih, M.-F. Chang, G. Lallement, B. Murmann, S. Mitra, and P. Raina, "Chimera: A 0.92-tops, 2.2-tops/w edge ai accelerator with 2-mbyte on-chip foundry resistive ram for efficient training and inference," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 4, pp. 1013–1026, 2022.

[13] F. Tu, Y. Wang, Z. Wu, W. Wu, L. Liu, Y. Hu, S. Wei, and S. Yin, "16.4 tensorcim: A 28nm 3.7 nj/gather and 8.3 tflops/w fp32 digital-cim tensor processor for mcm-cim-based beyond-nn acceleration," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 254–256.

[14] K. Prabhu, R. M. Radway, Y. Jeffrey, K. Bartolone, M. Giordano, F. Peddinghaus, Y. Urman, W.-S. Khwa, Y.-D. Chih, M.-F. Chang *et al.*, "Minotaur: An edge transformer inference and training accelerator with 12 mbytes on-chip resistive ram and fine-grained spatiotemporal power gating," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2024, pp. 1–2.

[15] A. Graening, S. Pal, and P. Gupta, "Chiplets: How small is too small?" in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[16] Y. Feng and K. Ma, "Chiplet actuary: a quantitative cost model and multi-chiplet architecture exploration," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 121–126. [Online]. Available: https://doi.org/10.1145/3489517.3530428

[17] M. Garcia Bardon, P. Wuytens, L.- Ragnarsson, G. Mirabelli, D. Jang, G. Willems, A. Mallik, A. Spessot, J. Ryckaert, and B. Parvais, "Dtco including sustainability: Power-performance-area-cost-environmental score (ppace) analysis for logic technologies," in *2020 IEEE International Electron Devices Meeting (IEDM)*, 2020, pp. 41.4.1–41.4.4.

[18] C.-C. Chou, Z.-J. Lin, P.-L. Tseng, C.-F. Li, C.-Y. Chang, W.-C. Chen, Y.-D. Chih, and T.-Y. J. Chang, "An n40 256k× 44 embedded rram macro with sl-precharge sa and low-voltage current limiter to improve read and write performance," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 478–480.

[19] R. Giterman, A. Shalom, A. Burg, A. Fish, and A. Teman, "A 1-mbit fully logic-compatible 3t gain-cell embedded dram in 16-nm finfet," *IEEE Solid-State Circuits Letters*, vol. 3, pp. 110–113, 2020.

[20] C.-H. Lin, B. Greene, S. Narasimha, and e. a. Cai, J., "High performance 14nm soi finfet cmos technology with 0.0174μm2 embedded dram and 15 levels of cu metallization," in *2014 IEEE International Electron Devices Meeting*, 2014, pp. 3.8.1–3.8.3.

[21] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Austin, TX, USA: IEEE, Apr. 2013, pp. 86–96.

[22] X. Bai, J. Zhao, S. Zuo, and Y. Zhou, "A 2.5 Gbps, 10-lane, low-power, LVDS transceiver in 28 nm CMOS technology," *Electronics*, vol. 8, no. 3, p. 350, 2019.

[23] Y. Zhang, X. Zhang, and M. S. Bakir, "Benchmarking digital die-to-die channels in 2.5-d and 3-d heterogeneous integration platforms," *IEEE Transactions on Electron Devices*, vol. 65, no. 12, pp. 5460–5467, 2018.

[24] M.-S. Lin, T.-C. Huang, C.-C. Tsai, K.-H. Tam, K. C.-H. Hsieh, C.-F. Chen, W.-H. Huang, C.-W. Hu, Y.-C. Chen, S. K. Goel, C.-M. Fu, S. Rusu, C.-C. Li, S.-Y. Yang, M. Wong, S.-C. Yang, and F. Lee, "A 7-nm 4-ghz arm¹-core-based cowos¹ chiplet design for high-performance computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 956–966, 2020.

[25] X. Sun, X. Peng, S. Q. Zhang, J. Gomez, W.-S. Khwa, S. S. Sarwar, Z. Li, W. Cao, Z. Wang, C. Liu *et al.*, "Estimating power, performance, and area for on-sensor deployment of ar/vr workloads using an analytical framework," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 6, pp. 1–27, 2024.

[26] S. W. Liang, G. C. Y. Wu, K. C. Yee, C. T. Wang, J. J. Cui, and D. C. H. Yu, "High performance and energy efficient computing with advanced soic™ scaling," in *2022 IEEE 72nd Electronic Components and Technology Conference (ECTC)*, 2022, pp. 1090–1094.

[27] C. Hu, M. Chen, W. Chiou, and C. Doug, "3d multi-chip integration with system on integrated chips (soic™)," in *2019 Symposium on VLSI Technology*. IEEE, 2019, pp. T20–T21.

ARCANE: Adaptive RISC-V Cache Architecture for Near-memory Extensions

Vincenzo Petrolo* Flavia Guella* Michele Caon* Pasquale Davide Schiavone† Guido Masera* Maurizio Martina*

*VLSI Lab
Politecnico di Torino, Italy
{vincenzo.petrolo,flavia.guella,michele.caon,guido.masera,maurizio.martina}@polito.it

†Embedded Systems Laboratory
EPFL, Switzerland
davide.schiavone@epfl.ch

***Abstract*—Modern data-driven applications expose limitations of von Neumann architectures—extensive data movement, low throughput, and poor energy efficiency. Accelerators improve performance but lack flexibility and require data transfers. Existing compute in- and near-memory solutions mitigate these issues but face usability challenges due to data placement constraints. We propose a novel cache architecture that doubles as a tightly-coupled compute-near-memory coprocessor. Our RISC-V cache controller executes custom instructions from the host CPU using vector operations dispatched to near-memory vector processing units within the cache memory subsystem. This architecture abstracts memory synchronization and data mapping from application software while offering software-based Instruction Set Architecture extensibility. Our implementation shows $30\times$ to $84\times$ performance improvement when operating on 8-bit data over the same system with a traditional cache when executing a worst-case 32-bit CNN workload, with only $41.3\,\%$ area overhead.**

***Index Terms*—In-cache computing, custom ISA extensions, RISC-V, Edge Computing.**

I. Introduction

Modern computing systems are increasingly demanding higher performance and energy efficiency due to data-intensive workloads. Traditional von Neumann architectures struggle with scaling due to the so-called "memory wall" [1] bottleneck, impacting performance and energy efficiency. To overcome these limitations, alternative computing paradigms like Compute-In-Memory (CIM) have been explored. In-Memory Computing (IMC) integrates processing within the memory array, reducing data movement and enhancing performance and energy efficiency. However, it faces challenges with complex arithmetic operations and memory density. Near-Memory Computing (NMC), using conventional digital flows and commercial memory macros, offers better scalability and reliability compared to IMC. ARCANE, the architecture proposed in this work, builds upon the NMC paradigm, aiming to bridge the gap between the flexibility of Central Processing Unit (CPU)-based computing and the energy and performance benefits of CIM. It leverages a tightly coupled instruction offloading mechanism based on the OpenHW Group CORE-V-X-IF (CV-X-IF) coprocessor interface [2] to offload complex instructions to a RISC-V cache controller, which implements them as vector kernels exploiting the CIM-oriented custom Instruction Set Architecture (ISA) extension from [3]. In other words, the ARCANE cache system doubles as a tightly coupled coprocessor that abstracts complex in-cache computing operations inside a software-defined ISA that wraps the underlying custom vector ISA used to program efficient near-memory Vector Processing Units (VPUs) constituting the cache memory space. The major contributions of the ARCANE cache architecture are:

- Efficient operation offloading: ARCANE leverages a CPU-based controller and a software-defined ISA to abstract complex in-cache operations, that are exposed to the host CPU as complex instructions, offloaded through a coprocessor interface.
- Seamless integration with existing systems: ARCANE is designed as a drop-in replacement for the conventional on-chip Last Level Cache (LLC) of a host Microcontroller Unit (MCU), minimizing the integration effort and ensuring compatibility with existing platforms.
- High-performance computing: ARCANE's VPUs operate directly on data residing in the LLC, achieving $30\times$ to $80\times$ higher throughput compared to a CPU-based approach when executing a 3-channel convolutional layer on 8-bit data.

The rest of this paper is organized as follows: section II introduces relevant works from the literature; section III describes and motivates ARCANE's architecture; section IV details the customizable ISA and the runtime running in the cache controller; section V reports and discusses implementation and performance figures; finally section VI concludes the paper.

II. Related Works

CIM approaches embedding processing elements within cache hierarchies have the inherent benefit of eliminating data transfers between system memory and dedicated compute units. Static Random-Access Memory (SRAM)-based IMC solutions [4], [5], [6], [7] repurpose the SRAM blocks inside the cache as Single Instruction Multiple Data (SIMD) accelerators exploiting bit-line computing and achieving high area and energy efficiency at the cost of reduced memory density and limited operational flexibility compared to conventional, off-the-shelf SRAMs [8].

In contrast, NMC solutions place arithmetic units outside memory subarrays, leveraging denser, commercial SRAM arrays and conventional digital implementation flow for greater portability. These systems achieve higher throughput and energy efficiency by processing data close to memory, avoiding costly transfers via system interconnect. Typical NMC architectures rely on data-parallel execution units that operate on vectors [9] or matrices [10] to accelerate Multiply-and-Accumulate (MAC) operations in neural network inference.

Software integration challenges limit CIM commercial diffusion [11]. IMC systems encode instructions for the memory as conventional bus transactions [4], thus requiring the application software or a dedicated compiler to generate the necessary commands. A more streamlined software integration can be achieved by connecting the CIM device to the host CPU through a dedicated instruction offloading interface similar to a coprocessor [9]. This approach facilitates the insertion of compute instructions into the generated

V. Petrolo and F. Guella contributed equally to this work.

application code, and guarantees synchronization with other in-flight instructions, implicitly handled by the CPU.

The architecture proposed in this work builds upon the NMC integration paradigm proposed with NM-Carus in [3] and the instruction-level offloading mechanism used in [9] to relieve the application software from explicitly handling memory management and synchronization with the cache.

III. Architecture

The proposed in-cache computing paradigm is designed to address the challenges of high-latency memory access in data-intensive applications by enabling computational functionality directly within the LLC. In this context, ARCANE offers two primary advantages: 1) it significantly reduces the impact of long latencies operation on the last memory level, effectively making them transparent to the system, 2) it facilitates efficient execution of data-intensive tasks by leveraging cache resources, allowing Out-of-Order (OoO) program execution and improved overall system throughput.

ARCANE (Figure 1) replaces a traditional data memory subsystem of a low-power MCU with a smart LLC that operates both as a cache and computational unit. It is connected to the system bus with two slave ports, as well as ports towards external memories (e.g., flash or pseudo-static RAMs (PSTRAMs)). In addition, the host CPU can offload complex custom RISC-V extensions for in-LLC operations that process cache data via the CV-X-IF [2]. Thus, from the host CPU point of view, the proposed LLC operates both as a cache and as an external co-processor. Computation within the LLC restricts the active cache region, introducing additional control complexity and potentially affecting performance. However, in-cache computing can partially offset the delays caused by long-latency complex operations, thanks to (2). Inside ARCANE, to mitigate potential performance degradation, the LLC controller ensures efficient cache resources management, ensuring a balance between computational throughput and memory coherence. Computation in the ARCANE LLC is enabled by leveraging the NMC paradigm, as presented in [3]. The custom in-LLC extensions are based on micro-programs built on top of the vector-like custom extensions proposed in the NMC IP called NM-Carus [3]. Vector-like instructions are chosen to limit the overhead associated with control flow instructions and to utilize SIMD capabilities to handle parallel workloads. Differently from the work proposed in [3], multiple NM-Carus instances managed by a single embedded CPU (eCPU) are employed to build the LLC system. The eCPU is based on the OpenHW Group's CV32E40X, a 4-stage, in-order RISC-V core based on [12], that offloads the custom vector-like near-memory operations to each NM-Carus instance, which act as VPUs. A dispatcher carries out the distribution to the selected VPUs, keeping the architecture modular and scalable. A slave port to the system bus facilitates the interaction between the host CPU and the embedded Memory (eMEM) to upload the eCPU firmware and configure memory-mapped registers, ensuring flexibility in deployment and programmability.

The architecture of ARCANE is analyzed in detail in Section III-A, focusing on its standard and in-cache computing functionalities. Furthermore, the offloading mechanism from the host CPU to the eCPU is discussed in Section III-B, introducing the concept of software-decoded instructions, which further boosts the system's flexibility and usability.

A. ARCANE LLC

1) Cache normal functioning mode: The LLC is designed as a fully associative cache, with a total number of lines equal to the aggregate vector register capacity of the system (i.e., the number of VPUs by vector registers per VPU). The cache line length is configured to match the maximum supported vector size to streamline memory management and ensure coherence between computational and caching operations, avoiding memory fragmentation issues. Cache hits are resolved in a single cycle, while misses and write-backs are handled by a dedicated Direct Memory Access (DMA). The LLC implements an approximate version of the Least Recently Used (LRU) replacement policy using a counter-based approach to maintain effective cache utilization. Furthermore, a write-back writing policy is enforced for improved performance, writing a cache line back to its original memory location on dirty replacement.

2) Cache locking and hazards management: In-cache computation enables parallel execution of the host CPU and the eCPU program flow. The cache controller must mediate between the Cache Runtime (C-RT) and the host CPU to handle contention on the VPUs. A locking mechanism is implemented through a memory-mapped configuration register, written by the eCPU and read by the controller to synchronize cache accesses. When the eCPU acquires the lock, the host CPU is blocked from accessing the cache until the lock is released. Conversely, a potential eCPU's lock request is not granted during ongoing host CPU operations, thus stalling the C-RT until the memory operation concludes.

During kernel execution, cache regions allocated to kernel operands are marked with a `busy computing` status to prevent access by normal operations, thus ensuring cache consistency while enabling in-cache computing. Potential hazards may arise from concurrent host CPU and eCPU operations. Write-After-Read (WAR) hazards occur when the host CPU issues a store operation on a source operand of an active kernel, potentially overwriting it before allocation is complete. As kernel operand allocation involves creating temporary copies in the VPU cache lines arranged according to the kernel layout, a blocking mechanism must prevent store operations on the sources until allocation is finalized. Read-After-Write (RAW) hazards arise if the host CPU reads the kernel result before the computation is complete, while Write-After-Writes (WAWs) emerge if the kernel destination overwrites the result of a subsequent store. All operations targeting kernel destinations must be blocked until kernel write-back is completed to avoid conflicts. The controller overcomes stalling conditions once memory contention resolves.

3) Address Table (AT): The system employs this auxiliary table to manage kernel source and destination states, ensuring proper synchronization and preventing data corruption while maintaining high throughput for in-cache computation. Each AT entry contains the start and end addresses of the operands, along with a validity and a status flag. The eCPU updates the AT when matrices are registered and sets their status to `busy` according to the hazard-avoidance policy. Additional status bits in the Cache Table (CT) indicate whether a cache line contains a source or a destination to streamline access. This approach allows the AT to be checked only when a corresponding cache line is marked as a source or destination, keeping a one-cycle delay in case of a hit. Cache misses always involve an AT lookup to determine if the request pertains to an allocated operand. If the required element belongs to a critical cache line but is not part of an operand, the entire line is loaded from memory, marked as containing an operand data and the request is served. This mechanism preserves program correctness by stalling only critical memory requests while allowing non-critical operations to proceed.

4) Software-Driven DMA: During the kernel allocation phase, the eCPU leverages X-HEEP's specialized DMA supporting 2D

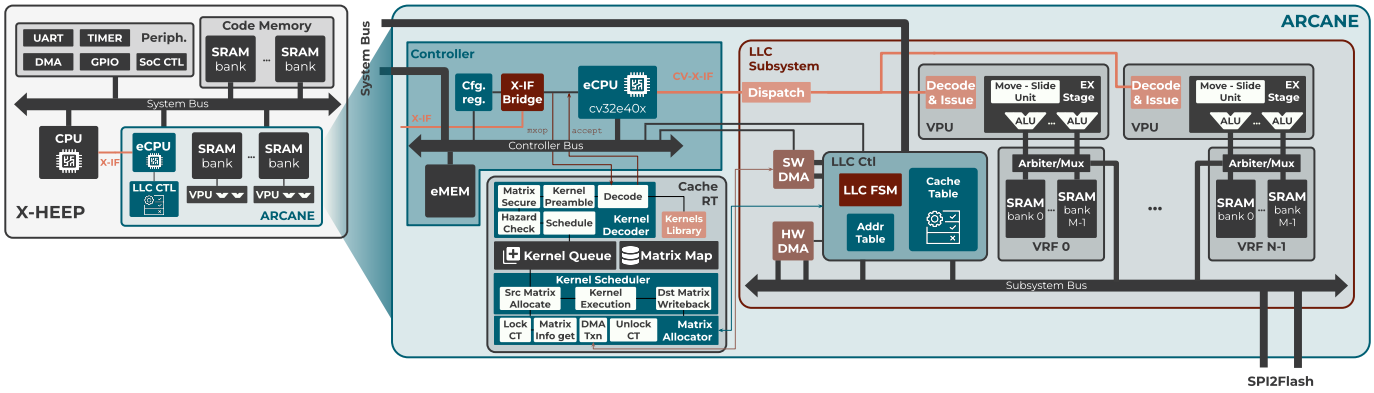


Figure 1: X-HEEP system level block diagram with a detailed view of the ARCANE LLC and software stack.

transactions, transferring operands from the main memory to the selected VPU in the required matrix format. To optimize latency, the LLC controller updates cache entry statuses upon receiving a DMA request, bypassing the need for software to search the CT. The DMA requests, as depicted in Figure 1, are routed through the LLC controller, which forwards data either from the cache (on a hit) or from the external off-chip memory (on a miss). The controller updates hit lines associated with sources or destinations and marks the lines required for computing as `busy computing`, handling write-back if required. During the kernel write-back, the cache follows a *fetch-on-write* policy, updating destinations directly in the cache and marking the corresponding lines as `dirty`.

B. Bridge

The bridge shown in Figure 1 provides a unified interface between the host CPU and eCPU, enabling offloading of matrix operations via the CV-X-IF. Operations are processed by the eCPU through an interrupt-driven, memory-mapped mechanism, completely transparent to the host CPU. The bridge samples the instruction's `opcode`, `func5`, and register operands coming from the offloaded RISC-V instruction over the CV-X-IF, making data accessible to the eCPU. Upon offloading, the bridge raises an interrupt for the eCPU to decode the instruction in SW. A dedicated eCPU register logs the decoding outcome, which the bridge forwards to the host CPU using CV-X-IF. The host CPU then sends commit or kill signals through the bridge, which idles upon kill acknowledgment. For operations proceeding to execution, the bridge notifies the host CPU, allowing it to continue the application workflow in an OoO fashion.

IV. Software-defined In-Cache Instruction Set Extensions

Despite its advantages, the widespread adoption of NMC in data-driven applications (e.g., neural networks, signal processing) is often limited by the complexity programmers face when writing kernels. This process requires a deep understanding of the hardware architecture and the precise data layout within the memory hierarchy. These challenges can be mitigated by encapsulating a kernel into a single complex instruction, thereby offloading data management to the existing CPU controller within the NMC subsystem. The CPU controller decodes the complex in-cache instructions in software, offering enhanced flexibility with negligible impact on application throughput, as demonstrated in Section V-B. The kernel itself, which represents the micro-program that executes the complex in-cache instruction, is built by leveraging the custom near-memory vector-like RISC-V extensions presented in [3], which are instead decoded and executed in HW by the NMC instances. This hierarchical approach makes programmability straight-and-forwards from the host CPU point of view, which handles compact and well-defined complex in-cache instructions while offering at the same time flexibility thanks to the SW-defined instructions, and performance, thanks to the vector-like near-memory instructions implemented with parallel data-path.

The following section introduces a reconfigurable in-cache SW-defined RISC-V matrix ISA that abstracts the complexity of NMC. It is followed by a detailed discussion of the software system residing within the NMC subsystem.

A. Extendable In-Cache Matrix ISA

The `xmnmc` extension is implemented within the RISC-V *Custom-2* 25-bit encoding space, utilizing the `0x5b` major opcode. To maximize the utility of a single instruction, each source register is divided into 16-bit pairs, with four registers allocated for matrix register indices and two reserved for scalar parameters, α and β. This configuration accommodates parameter-intensive kernels, such as General Matrix Multiplication (GeMM), while maintaining flexibility. To ensure a high level of abstraction, the extension includes only two types of instructions:

1) Matrix Reserve (`xmr`): The `xmr` instruction binds a matrix's memory address and shape to a logical matrix register. Leveraging software-hardware cooperation within the LLC subsystem, memory coherency is ensured with no additional effort from the programmer. Unlike a matrix load instruction as proposed by the T-HEAD RVM proposal [13], `xmr` does not immediately load matrix data into memory. Instead, it establishes a binding, deferring the memory load to a later stage, such as when it is explicitly requested by a matrix kernel operation. This deferred approach abstracts memory management complexities, significantly reducing the programmer's workload.

2) Matrix Kernels (`xmkN`): Matrix kernel instructions, denoted as xmkN where N∈[0,30], define up to 31 distinct complex matrix kernel operations. The func5 field within the `0x5b` opcode specifies the kernel operation. This flexibility is further enhanced by the reprogrammable software decoder, allowing for updates, and further ISA extensions.

Internally, kernels represent matrices as groups of vector registers, maximizing the reuse of the custom vector-like extension provided by the chosen NMC architecture [3]. Currently, five complex matrix kernels have been implemented, some of which inherited from existing implementations [3].

Mnemonic	Data sources						Description
	hi(rs1)	lo(rs1)	hi(rs2)	lo(rs2)	hi(rs3)	lo(rs3)	
xmr.[w,h,b]	hi(&A)	lo(&A)	A.stride	md	A.cols	A.rows	Matrix reserve
xmk0.[w,h,b]	α	β	ms3	md	ms1	ms2	GeMM
xmk1.[w,h,b]	α	–	–	md	ms1	–	LeakyReLU
xmk2.[w,h,b]	stride	win_size	–	md	ms1	–	Maxpooling
xmk3.[w,h,b]	–	–	–	md	ms1	ms2	2D Conv.
xmk4.[w,h,b]	–	–	–	md	ms1	ms2	3-ch. 2D Conv.

Table I: Example of ARCANE custom kernels.

```
// Convolution Layer
int main(void) {
  int A[rowsA][colsA] = {...}
  ...
  // Reservation
  _xmr_w(m0, A, 1, rowsA, colsA);
  _xmr_w(m1, F, 1, rowsF, colsF);
  _xmr_w(m2, R, 1, rowsR, colsR);
  // Matrix Kernel
  _conv_layer_w(m2, m0, m1);
  ...
```

Listing 1: xmnmc application example.

To demonstrate the abstraction capabilities of the xmnmc extension, a 3-channel 2D convolution kernel operation inspired by ImageNet has been implemented. This kernel integrates 2D convolution, max-pooling, and ReLU activation while supporting matrices of arbitrary dimensions.

An example of using the xmnmc is provided in Listing 1.

B. Cache Runtime System

C-RT is a lightweight runtime system designed to perform three core tasks: software decoding of matrix operations, their scheduling and execution, and matrix allocation. It is executed only by the eCPU within the LLC. These modules operate independently but can communicate. C-RT operates as a single-threaded, preemptive runtime, ensuring efficient handling of offloaded matrix operations, even during kernel execution. It follows a producer-consumer model centered around a statically allocated kernels queue.

C-RT employs a static memory allocation philosophy, offering two key benefits: predictable runtime without memory fragmentation and the ability to analyze maximal stack usage. Critical structures, such as the kernel queue and the matrix map, are preallocated to fixed sizes determined by system configuration. For instance, the matrix map supports a configurable number of logical matrix registers. A user-configurable kernel library allows custom kernels to be added before C-RT compilation. Additionally, C-RT supports a deep-sleep mode for power efficiency when no operations are pending.

The primary modules of C-RT, are the Kernel Decoder, Kernel Scheduler, and Matrix Allocator:

1) Kernel Decoder: Operates within the interrupt handler, decoding matrix operations offloaded by the host CPU. It retrieves kernel information, such as preambles and function addresses, using the operation's opcode and func5 field with $O(1)$ complexity access to the kernel library. If the operation is recognized, the kernel preamble is executed, and upon success, the operation is scheduled and added to the kernel queue. Due to ooo communication with the host CPU, data-inbound matrices may risk modification before subsequent computations. To address this, the Kernel Decoder records the start and end addresses of the memory region in the AT, preventing undesired reads/writes to a destination/source operand still required by pending matrix operations. Additionally, to mitigate hazards such as having a xmr overwriting an older reservation still in use, the Kernel Decoder employs a hazard checker that internally renames logical matrices effectively solving the hazard. Notably, matrix allocation does not happen during xmr execution. Instead, it is deferred until explicitly required by a kernel operation, enabling kernel-dependent layout optimization.

2) Kernel Scheduler: Manages the execution of matrix operations. Before execution, it selects an appropriate VPU based on a policy that prioritizes VPUs with the fewest dirty cache lines. Once a VPU is selected, the scheduler invokes the Matrix Allocator to prepare source matrices with the required layout. After kernel execution, the scheduler determines whether the destination matrix will serve as a source operand in future operations. If not, it writes the matrix back to memory using the Matrix Allocator Application Programming Interface (API).

3) Matrix Allocator: Handles memory management for matrix operations. It receives the matrix operand layout information to program 2D DMA transfers that move data from memory to the selected VPU. To prevent clashes in cache line access with the host CPU, the allocator must first acquire a lock on the cache controller. The Matrix Allocator further minimizes the overhead impact on throughput by allocating the effective dimensions of the matrix. A custom DMA controller ensures data integrity by detecting writes on dirty cache lines and triggering writebacks. After the allocation process is completed, the Matrix Allocator releases the cache controller lock. During the writeback phase, which occurs post-kernel execution, the Matrix Allocator locks the LLC Controller and programs a 2D DMA transfer to consolidate scattered matrix-shaped data into a contiguous array inside the LLC. If the cache line containing the matrix is not already present in the LLC, it is first loaded and then updated with the newly computed data from the matrix operation. This ensures that any pending access requests for the updated data can be promptly served with the latest data. Once the transfer completes, it marks the previously busy cache lines as free and releases the LLC Controller and memory region. The memory region is then made accessible to the host CPU by modifying the AT's access permissions.

V. Experimental Results

A. Logic Synthesis

The logic synthesis of ARCANE is performed using Synopsys Design Compiler® 2020.09, targeting a low-power 65nm LP CMOS technology library under worst-case operating conditions with a target clock frequency of 250 MHz. ARCANE is encapsulated within the eXtendible Heterogeneous Energy-Efficient Platform (X-HEEP) MCU framework replacing the conventional data memory. Three configurations, providing a range of design trade-offs to balance computational throughput and area overhead, are synthesized, differing in the number of lanes, and, consequently, the number of memory banks per VPU. Across all configurations, the cv32e40x core implementing the RV32IMC instruction set serves as the eCPU, supported by a 16 kiB eMEM. The MCU instruction memory consists of 4 banks of 32 kiB each, for a total of 128 kiB, while the data cache has a total capacity of 128 kiB and is split into 4 VPUs with a vector length and a cache line size of 1 kiB. The X-HEEP system, featuring the cv32e40px core [12], with identical instruction memory size and augmented with a standard data LLC, is used as a baseline for a fair comparison. Table II compares the synthesized configurations, highlighting their area overhead with respect to the baseline. The

additional area introduced by ARCANE primarily stems from its enhanced computational capabilities, as shown in Figure 2, while the impact of the additional control logic for managing concurrent computation and memory operations remains negligible. For the intermediate 4-lanes configuration, ARCANE shows a 28.3% area increase over the baseline, with 22% attributed to the vector pipelines and 5% to the controller, split among the eCPU and the eMEM. As the number of lanes increases, area overhead grows due to more complex computation logic, reduced memory density from LLC division, and higher routing complexity. Notably, the additional cache control logic accounts for less than 4% of the total system area.

As a final remark, ARCANE LLC does not increase the critical path of the target frequency of 250 MHz.

Table II: Synthesis results with 16 KiB eMEM.

Conf	ARCANE (4 VPUs[a], 2 lanes)	ARCANE (4 VPUs[a], 4 lanes)	ARCANE (4 VPUs[a], 8 lanes)	X-HEEP[14] (4 DMem Banks[a])
Area[μm^2]	2.88×10^6 (+21.7%)	3.03×10^6 (+28.3%)	3.34×10^6 (+41.3%)	2.36×10^6
Area[kGE][c]	1996	2105	2318	1640

[a] 32 KiB each. [c] GE is the 2-input drive strength-one NAND gate equivalent area.

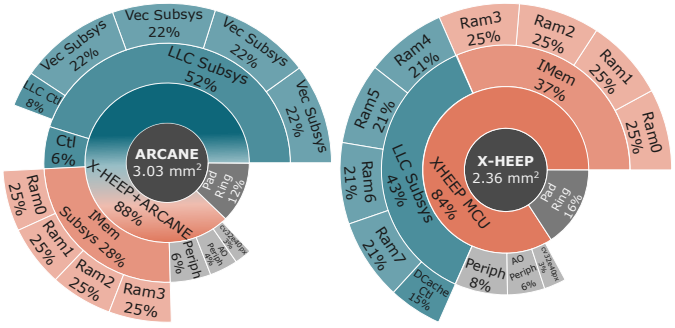


Figure 2: Area split of X-HEEP + ARCANE 4-lanes configuration (128 kiB) versus X-HEEP + standard data LLC (128 kiB)

B. Overhead Analysis

Providing abstraction to programmers is a primary goal of the `xmnmc` extension, achieved through software decoding, allocation, kernel execution, and writeback phases. In Listing 1, multiple `xmr` instructions define kernel operands in the preamble phase, deferring data loading to kernel execution. Data movement during allocation and writeback is handled by DMA transfers. These phases introduce throughput overhead. A worst-case study of a 3-channel 2D convolution kernel with 3×3 filters on `int32` integers was conducted using 2-, 4-, and 8-lanes ARCANE shown in Figure 3. Preamble phase overhead decreases exponentially with input size, from 60% for small inputs to 2.89% for larger ones, making ARCANE a suitable solution for relatively large input sizes. Allocation overhead grows with lane count, saturating globally at 15%, proportional to the input size. Writeback overhead falls linearly with input size, reaching 2% for the largest matrices. As inputs increase the compute phase dominates as input size increases, with overhead saturating at 20% under worst-case conditions.

C. Comparison with the State of the Art

We compare ARCANE with the state-of-the-art (Section II), including a baseline CV32E40X CPU core for speedup measurement and the CV32E40PX [12], a core implementing the XCVPULP ISA extensions for 8- and 16-bit data, supported by the OpenHW Group toolchain. We use a 3-channel 2D convolution layer (Listing 1),

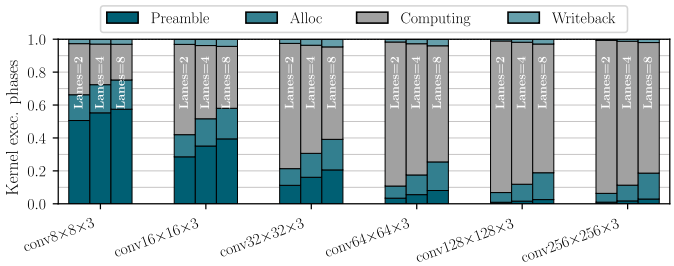


Figure 3: Non-compute phases overhead analysis under different input matrix sizes and ARCANE lanes with `int32` datatype.

relevant to edge AI and tiny CNNs, for comparison with varying filter sizes, ARCANE lane configurations, and data types. Results (Figure 4) show that ARCANE's 2-lane configuration achieves peak throughput at 64×64 inputs, saturating faster with larger filters due to lane limits. In `int8`, prevalent in tinyML, performance saturation occurs in 2-lane setups, while 4- and 8-lane setups show consistent speedups with larger inputs due to optimized DMA transfers reducing allocation times. Although CV32E40PX outperforms ARCANE at smaller input sizes, its scaling peaks at $8.6\times$ due to overhead from repeated data loading. ARCANE's overhead is higher for smaller inputs but excels in processing large datasets. For instance, at 256×256 inputs with `int8` 3×3 filters, ARCANE's 8-lane setup achieves a $30\times$ speedup over CV32E40X, compared to CV32E40PX's $5\times$. In multi-instance mode with 4 VPUs and 8 lanes, ARCANE achieves a $120\times$ speedup compared to CV32E40X and $1.6\times$ compared to CV32E40PX, with area utilization comparable to a 15-core CV32E40PX system, excluding logic and bus contributions. Multi-core implementations relying on packed-SIMD instructions introduce significant overhead from frequent instruction cache accesses, causing memory contention and synchronization delays. Even under optimal conditions, the theoretical speedup peaks at $75\times$, far below ARCANE's.

As motivated in Section II, BLADE [4] and Intel CNC [9] are selected as candidates for comparisons. Due to their restricted set of supported operations, running a direct comparison employing the 3-channel 2D convolution layer is not feasible. Nevertheless, it is possible to compare their peak throughputs. Given the difference in the technological nodes of such solutions, the results are scaled using an operational clock frequency of 330 MHz, typical value of an embedded 32 KiB SRAM in the 65 nm node. BLADE [4] implements a NMC architecture with a scaled area of 580×10^3 μm^2, making it $3.18\times$ smaller than ARCANE. However, its peak throughput is limited to 5.3 GOPS[1], while ARCANE, running at 265 MHz, achieves a peak throughput of 17.0 GOPS —an improvement of approximately $3.2\times$. This translates to an area efficiency of 9.1 GOPS/mm^2 for BLADE, compared to 9.2 GOPS/mm^2 for ARCANE, which demonstrates slightly superior area efficiency and also supports an extensible ISA while BLADE's is restricted to basic arithmetic operations. Intel CNC [9], fabricated using Intel's 4 technology node, renders scaled area comparisons impractical due to substantially different fabrication technologies. Nevertheless, its reported area is 1920×10^3 μm^2, larger than that of ARCANE in 65 nm node. The peak throughput of Intel CNC is 25.0 GOPS, achieving a $1.47\times$ speedup compared to ARCANE. Despite this, ARCANE still demonstrates greater programming flexibility, as Intel CNC supports only the MAC operation.

[1]One MAC operation is considered as two OP (one multiplication and one addition), as commonly done in literature.

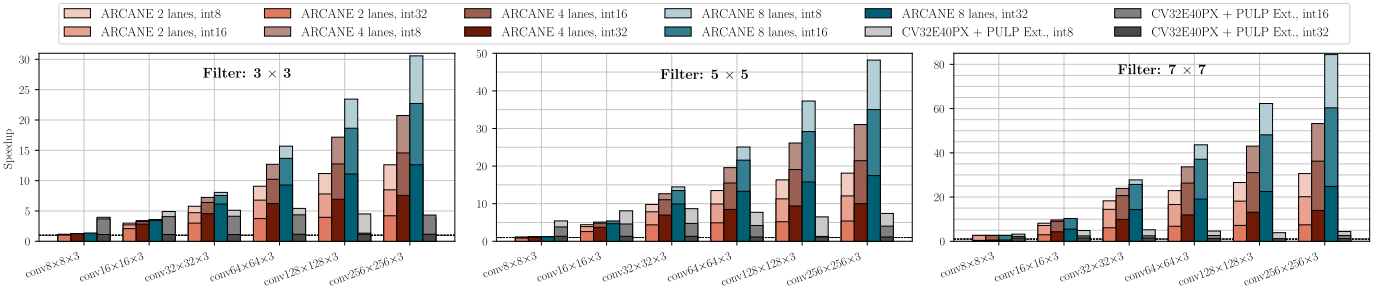


Figure 4: Speedup comparison between single instance ARCANE configurations, CV32E40X and CV32E40PX featuring XCVPULP extensions.

VI. Conclusion & Future Works

The ARCANE in-cache computing architecture combines the energy and performance benefits of the NMC paradigm with the programmability of a CPU-based solution. It does so by leveraging a RISC-V cache controller and a customizable, software-defined ISA that operates as an abstraction layer to offload complex matrix operations to the cache. ARCANE is a drop-in replacement for a system's LLC, doubling as a programmable matrix coprocessor. The cache controller relieves the application software from explicitly managing data movement and synchronization, enabling straightforward software development. When executing an 8-bit $256 \times 256 \times 3$ convolutional layer with a 7×7 filter, ARCANE achieves a performance improvement of $84\times$ over a scalar CPU implementation (RV32IIMC) and $16\times$ over the XCVPULP packed-SIMD and DSP-enhanced ISA. It achieves comparable performance to existing solutions with a low area overhead of $41.3\,\%$ when integrated into an edge-oriented MCU.

Acknowledgments

This work is supported by the EU TRISTAN project with GA 101095947, which has received funding from the CHIPS Joint Undertaking and its members, and including top-up funding by *Ministero dello sviluppo economico*, and SERICS (PE00000014), under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

[1] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and memory wall," *IEEE Micro*, vol. 44, no. 3, pp. 33–39, 2024.

[2] OpenHW Group, "Openhw group specification: Core-v extension interface (cv-x-if)," online, 2023, accessed: Nov 19, 2024. [Online]. Available: https://github.com/openhwgroup/core-v-xif

[3] M. Caon, C. Choné, P. D. Schiavone, A. Levisse, G. Masera, M. Martina, and D. Atienza, "Scalable and RISC-V programmable near-memory computing architectures for edge nodes," 2024. [Online]. Available: https://arxiv.org/abs/2406.14263

[4] W. A. Simon, Y. M. Qureshi, M. Rios, A. Levisse, M. Zapater, and D. Atienza, "BLADE: An in-cache computing architecture for edge devices," *IEEE Transactions on Computers*, vol. 69, no. 9, pp. 1349–1363, 2020.

[5] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 481–492.

[6] R. Fan, Y. Cui, Q. Chen, M. Wang, Y. Zhang, W. Zheng, and Z. Li, "MAICC : A lightweight many-core architecture with in-cache computing for multi-DNN parallel inference," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 411–423.

[7] D. Fujiki, S. Mahlke, and R. Das, "Duality cache for data parallel acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. Association for Computing Machinery, 2019, p. 397–410. [Online]. Available: https://doi.org/10.1145/3307650.3322257

[8] Y. M. Qureshi, W. A. Simon, M. Zapater, K. Olcoz, and D. Atienza, "Gem5-X: A many-core heterogeneous simulation platform for architectural exploration and optimization," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, 2021. [Online]. Available: https://doi.org/10.1145/3461662

[9] G. K. Chen, P. C. Knag, C. Tokunaga, and R. K. Krishnamurthy, "An eight-core RISC-V processor with compute near last level cache in Intel 4 CMOS," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 1117–1128, 2023.

[10] A. V. Nori, R. Bera, S. Balachandran, J. Rakshit, O. J. Omer, A. Abuhatzera, B. Kuttanna, and S. Subramoney, "REDUCT: Keep it close, keep it cool! : Efficient scaling of DNN inference on multi-core CPUs with near-cache compute," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 167–180.

[11] A. A. Khan, J. P. C. D. Lima, H. Farzaneh, and J. Castrillon, "The landscape of compute-near-memory and compute-in-memory: A research and commercial overview," 2024.

[12] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with dsp extensions for scalable IoT endpoint devices," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[13] XUANTIE-RV, "RISC-V matrix multiplication extension specification v0.3.0," online, 2023, accessed: Nov 19, 2024. [Online]. Available: https://github.com/XUANTIE-RV/riscv-matrix-extension-spec/releases/tag/v0.3.0

[14] P. D. Schiavone, S. Machetti, M. Peón-Quirós, J. Miranda, B. Denkinger, T. C. Müller, R. Rodríguez, S. Nasturzio, and D. A. Alonso, "X-HEEP: An open-source, configurable and extendible RISC-V microcontroller," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, ser. CF '23. Association for Computing Machinery, 2023, p. 379–380. [Online]. Available: https://doi.org/10.1145/3587135.3591431

zkVC: Fast Zero-Knowledge Proof for Private and Verifiable Computing

Yancheng Zhang[†], Mengxin Zheng[†], Xun Chen[‡]
Jingtong Hu[§], Weidong Shi[††], Lei Ju[‡‡], Yan Solihin[†], Qian Lou[†*]
[†]University of Central Florida, [‡]Samsung Research America,
[§]University of Pittsburgh, [††]University of Houston, [‡‡]Shandong University

***Abstract*—In the context of cloud computing, services are held on cloud servers, where the clients send their data to the server and obtain the results returned by server. However, the computation, data and results are prone to tampering due to the vulnerabilities on the server side. Thus, verifying the integrity of computation is important in the client-server setting. The cryptographic method known as Zero-Knowledge Proof (ZKP) is renowned for facilitating private and verifiable computing. ZKP allows the client to validate that the results from the server are computed correctly without violating the privacy of the server's intellectual property. Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARKs), in particular, has been widely applied in various applications like blockchain and verifiable machine learning. Despite their popularity, existing zkSNARKs approaches remain highly computationally intensive. For instance, even basic operations like matrix multiplication require an extensive number of constraints, resulting in significant overhead. In addressing this challenge, we introduce *zkVC*, which optimizes the ZKP computation for matrix multiplication, enabling rapid proof generation on the server side and efficient verification on the client side. zkVC integrates optimized ZKP modules, such as Constraint-reduced Polynomial Circuit (CRPC) and Prefix-Sum Query (PSQ), collectively yielding a more than 12-fold increase in proof speed over prior methods. The code is available at https://github.com/UCF-Lou-Lab-PET/zkformer.**

***Index Terms*—Private and Verifiable Computing, Zero-Knowledge Proof, Machine Learning**

I. Introduction

Zero-knowledge proof (ZKP) [1], [2] is a cryptographic primitive that enables a prover to convince a verifier of the correctness of a computation without revealing the prover's secret input. ZKP ensures that a proof passes verification only if the computation was performed correctly, guaranteeing both verifiability and privacy. By providing strong guarantees on the integrity of computations while preserving the privacy of the server's input, ZKP has found broad applications in domains where computational integrity is critical, such as blockchain and verifiable machine learning.

In Figure 1, we illustrate one use case of ZKP in verifiable machine learning. ZKP allows the owner of a proprietary machine learning model, *prover*, to prove to the users, *verifier*, that predictions have been accurately computed by the pre-trained model, without compromising the model's confidentiality. As depicted, the client-side verifier first sends input data X to the server, where the server's prover performs the neural network inference $f(X, W)$ to output a prediction. ZKP is then used to generate a proof for this inference, ensuring $f(X, W)$'s validity without revealing the model weights, enabling the client to verify the proof.

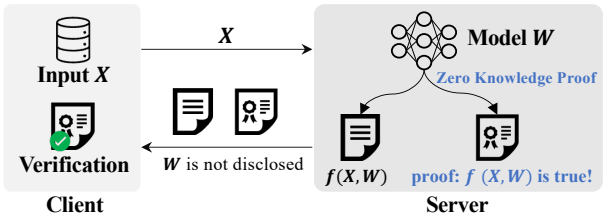


Fig. 1: Example of use case in verifiable and private neural network inference based on Zero-knowledge Proof (ZKP).

While promising, ZKP often leads to large computation overhead in practice, especially when proving matrix multiplication. Proving a single matrix multiplication of dimension $[49, 320] \times [320, 512]$ on a 16-Core CPU with a commonly used ZKP scheme [2] known for efficiency, can take up to ~ 3 minutes. Matrix multiplication is a fundamental operation in numerous applications, including data analysis and machine learning. However, the inefficiencies in current ZKP frameworks make it impractical to directly scale them for real-world scenarios involving massive matrix multiplications. For instance, Transformer-based models [3], [4], widely used in machine learning, rely heavily on extensive matrix multiplications. Proving the correctness of a single inference using a ViT-Base model [4] on the ImageNet dataset [5] results in prohibitively high computational costs, rendering such approaches infeasible.

A series of works have explored how to scale up ZKP-based verifiable computations for real-world applications [6]–[9], where many of them are built upon zk-SNARKs. zk-SNARKs is a branch of general ZKP, which features short proof sizes, fast verification, and non-interactivity and is suitable for cloud computing. The main challenge in reducing the overhead of zk-SNARKs is to reduce the number of constraints. Recent works [6], [9] reduce proving complexity by expressing convolution as polynomial multiplication within a polynomial quadratic arithmetic program (QAP). However, their optimization highly depends on the convolution's unique feature, which cannot directly extend to general matrix multiplication. The efficient

*Corresponding author: Qian Lou (qian.lou@ucf.edu)

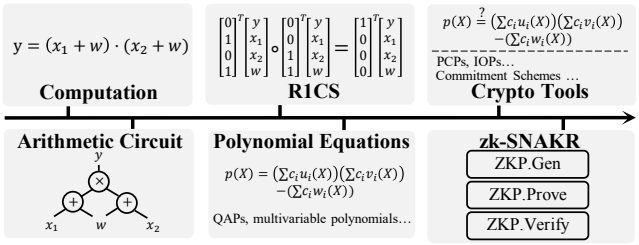


Fig. 2: The workflow of zero-knowledge proof systems.

construction of zk-SNARKs for matrix multiplication remains an open problem.

Proving the correctness of matrix multiplication begins by transforming it into an arithmetic circuit, which is then represented as a QAP. The efficiency of zk-SNARKs relies heavily on the complexity of the QAP, determined by two key factors: (1) the number of constraints, dictated by the multiplication gates in the arithmetic circuit, which also defines the degree of the constraint system, and (2) the number of variables, corresponding to the length of a full assignment to the circuit. Previous work [6] proposed reducing the number of constraints in convolution operations by introducing additional variables, referred to as dummy terms. However, directly applying this approach to matrix multiplication significantly increases the variable count, reducing overall efficiency. Notably, similar to the construction in vCNN [6], our verification process remains succinct and is largely independent of the complexity of the original computation.

Our Contributions. This paper introduces *zkVC*, an efficient zk-SNARK construction tailored for general operations such as matrix multiplication and their applications in machine learning, including attention-based Transformers. We propose **Constraint-Reduced Polynomial Circuits (CRPC)** to minimize the constraints required for matrix multiplication in ZKP. By transforming matrix multiplication into polynomial multiplication represented in a quadratic arithmetic program (QAP), CRPC reduces the number of constraints from $O(n^3)$ to $O(n)$. Additionally, we introduce **Prefix-Sum Query (PSQ)**, a technique that reduces the number of variables by optimizing the circuit for product accumulation in matrix multiplication. Experimental results demonstrate that the combination of CRPC and PSQ achieves a 12× improvement in proving time for matrix multiplication compared to prior methods. Furthermore, we apply *zkVC* to verifiable Transformer inference, achieving over a 15× runtime reduction on ViT models compared to baselines without our optimizations.

II. Background and Motivation

ZKP Systems. ZKP is a cryptographic protocol enabling a prover $\mathcal{P}$ to assure a verifier $\mathcal{V}$ of a statement's truth without disclosing anything beyond its validity. In verifiable neural networks, it lets a model owner $\mathcal{P}$ confirm the accuracy of a neural network inference to a client $\mathcal{V}$. Figure 2 outlines a ZKP system's process. To prove a computation like $y = (x_1 + w) \cdot (x_2 + w)$, it is first translated into an arithmetic circuit using addition and multiplication gates. This circuit is then converted into a constraint system, such as the Rank-1 Constraint System (R1CS) [1], which generalizes arithmetic circuit satisfiability. In R1CS, additions are represented by row vectors, for instance, $[0, 1, 0, 1]$ for $(x_1 + w)$, while multiplications are encoded using element-wise products. Therefore, proving the original computation's correctness equals satisfying the R1CS. Efficient R1CS checking involves encoding it into polynomials. Schemes like [1], [2] use Quadratic Arithmetic Programs (QAP) to encode R1CS, derived through polynomial interpolation on R1CS instances. Other approaches employ univariate polynomials [10], [11] or multivariate polynomials [12], [13] for encoding.

The verification of R1CS or its polynomial equations requires specific cryptographic tools like groth16 [2] or Spartan [13]. groth16, a widely-used ZKP construction, employs probabilistically checkable proofs (PCPs) and elliptic curve pairings for QAP verification. Spartan, on the other hand, uses Interactive Oracle Proof (IOP) [18] and polynomial commitment schemes [19] for multivariate polynomial checks, notably without requiring a trusted setup. A ZKP construction consists of three Probabilistic Polynomial Time (PPT) algorithms: Gen, Prove, and Verify. Gen creates public parameters from security parameters. Prove uses these parameters, public and private inputs, to produce a proof. Verify, using the public parameters, input, and proof, decides its validity. ZKP ensures proof completeness and soundness, meaning correct computations pass verification, and it is computationally hard for a dishonest prover to validate incorrect computations.

Homomorphic Encryption for Private Computation. Homomorphic encryption (HE) enables computations directly on encrypted data, removing the need for decryption and thus supporting privacy-preserving outsourcing of computa-

TABLE I: The comparison between zkVC and prior verifiable DNN methods including SafetyNets [14], Keuffer's [15], vCNN [6] , VeriML [16], ZEN [17] , zkCNN [7], zkML [8] and pvCNN [9].

Schemes	zk.	Non-Inter.	Const. Proof	No Trusted Setup	Trans-formers	Efficient MatMult	zk-ML Codesign
SatetyNets	X	X	X	✓	X	X	X
zkCNN	✓	X	X	✓	X	X	X
Keuffer's	✓	✓	✓	X	X	X	X
vCNN	✓	✓	✓	X	X	X	X
VeriML	✓	✓	✓	X	X	X	X
ZEN	✓	✓	✓	X	X	X	X
zkML	✓	✓	X	X	X	X	X
pvCNN	✓	✓	✓	X	X	X	X
zkVC	✓	✓	✓	✓	✓	✓	✓

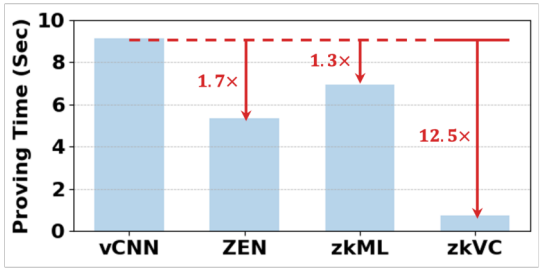
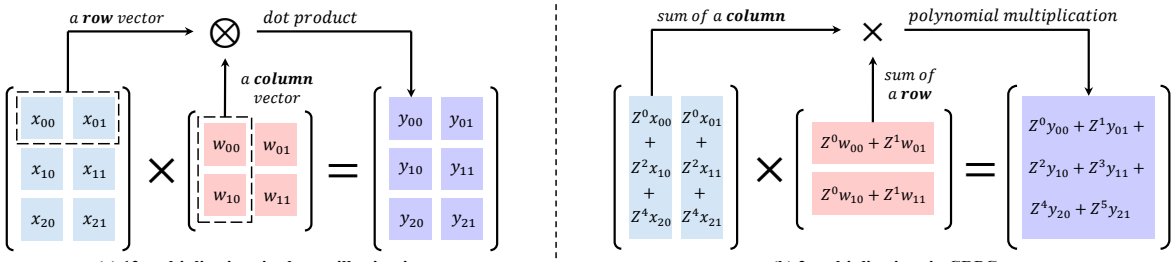


Fig. 3: Proving Time Comparison for Matrix Multiplication with Prior Work.



(a) 12 multiplications in the vanilla circuit

(b) 2 multiplications in CRPC

Fig. 4: Comparison of (a) basic constraint circuits with 12 multiplications and our (b) CRPC with 2 multiplications by transforming original matrix multiplication into polynomial multiplications of an intermediate variable Z.

tions [20]–[23]. However, HE alone does not inherently offer computational verifiability as ZKP do [24], [25]. Conversely, while ZKP ensures computational integrity, it lacks the data privacy guarantees provided by HE [26]–[29].

Comparison with Related Work. Table I shows the comparison of our zkVC and related works. SafetyNets [14] lacks zero-knowledge properties, leaving model weight privacy unprotected. Only SafetyNets and zkCNN [7] are interactive, necessitating ongoing communication between prover and verifier. Interactive ZKPs offer quicker proving times but require ongoing exchanges between the prover and verifier. While zero-knowledge polynomial commitment [30] and the Fiat-Shamir heuristic [31] could theoretically make these non-interactive, their security and efficiency impacts are unclear. The need for constant connectivity in interactive setups is a limitation, particularly for clients with limited hardware and power resources [17]. Table I's third column indicates that in SafetyNets, zkCNN, and zkML's [8], proof size grows logarithmically with model size, increasing verifier workload. Other schemes maintain a constant proof size. Non-interactive schemes generally require a trusted setup for public parameter generation, but interactive ones like SafetyNets and zkCNN do not. zkVC, using transparent zk-SNARKs such as Spartan, also bypasses the need for a trusted setup. Previous research concentrated solely on CNNs, while zkVC focuses on general matrix multiplication, introducing efficient modules for proving matrix multiplication.

Motivation. Matrix multiplication is essential in many applications but challenging to prove using ZKP. Figure 3 shows that in prior vCNN [6], proving a small matrix multiplication of dimension $[49, 64]\times[64, 128]$ takes as long as 9 seconds. Despite ZEN [17] introducing advanced quantization and zkML's [8] using a more efficient ZKP method, their improvements in matrix multiplication speed are still limited. This motivates us to design efficient ZKP modules for matrix multiplication. The proposed zkVC achieves a 12.5× reduction in proof time compared to the previous vCNN.

III. zkVC Design

A. Constraint-reduced Polynomial Circuits (CRPC)

Matrix multiplication plays a foundational role in various computational tasks, but its efficient representation in QAP remains challenging. As in Figure 4 (a), in vanilla QAP, every individual multiplication requires a distinct constraint. As every y_{ij} is a dot product between a row vector of X and a column vector of W, we have $y_{00} = x_{00} \cdot w_{00} + x_{01} \cdot w_{10}$, ... and $y_{21} = x_{20} \cdot w_{01} + x_{21} \cdot w_{11}$, where 12 multiplications result in 12 constraints in the QAP. Consider the matrix multiplication $Y = X \times W$ where $X \in \mathbb{R}^{a\times n}$, $W \in \mathbb{R}^{n\times b}$ and $Y \in \mathbb{R}^{a\times b}$. Our insight is that the products in $\{x_{ik} \cdot w_{kj}\}_{k=0}^{n-1}$ for y_{ij} can be encoded in polynomial multiplications. One intuitive transformation is:

$$\begin{aligned}(y_{00} + y_{01} + y_{10} + y_{11} + y_{20} + y_{21}) \\ = (x_{00} + x_{10} + x_{20}) \cdot (w_{00} + w_{01}) \\ + (x_{01} + x_{11} + x_{21}) \cdot (w_{10} + w_{11})\end{aligned}$$

When matrix multiplication is satisfied, the above equation is also satisfied. However, the converse is not guaranteed. It is possible that the sum of all y_{ij} is correct while individual y_{ij} is not. Essentially, this transformation ensures completeness but compromises the soundness required by zk-SNARKs.

As prior work's approach [6] suggests, it is possible to encode the matrix multiplication in one single polynomial multiplication, if the coefficients are properly arranged. Another possible transformation is:

$$\begin{aligned}(Z^1 y_{00} + Z^3 y_{01} + Z^5 y_{10} + Z^7 y_{11} + Z^9 y_{20} + Z^{11} y_{21}) \\ \neq (Z^1 x_{00} + Z^0 x_{01} + Z^5 x_{10} + Z^4 x_{11} + Z^9 x_{20} + Z^8 x_{21}) \\ \cdot (Z^0 w_{00} + Z^2 w_{01} + Z^1 w_{10} + Z^3 w_{11})\end{aligned}$$

where $y_{00} = x_{00} \cdot w_{00} + x_{01} \cdot w_{10}$ is incorporated within $Z^1 y_{00} = Z^1(x_{00} \cdot w_{00} + x_{01} \cdot w_{10})$. However, this transformation is not strictly equivalent. Firstly, the equation itself is not satisfied. The polynomial multiplication results in a lot of superfluous terms such as $Z^0 x_{01} w_{01}$, which does not belong to any y_{ij}. Secondly, to equate the two sides, all these dummy terms need to be included, which leads to increased number of variables and makes the circuit hard to prove.

While the above two transformations can reduce the number of multiplications, they cannot ensure the integrity of matrix multiplication. We propose CRPC to address this issue. Our insight is that the products in $\{x_{ik} \cdot w_{kj}\}_{k=0}^{n-1}$ for y_{ij} result only from the multiplications between elements in the k_{th} column in X and elements in k_{th} row in W. As shown in Figure 4 (b), we transform each column of X and each row of W into polynomials of a random intermediate variable Z, for example,

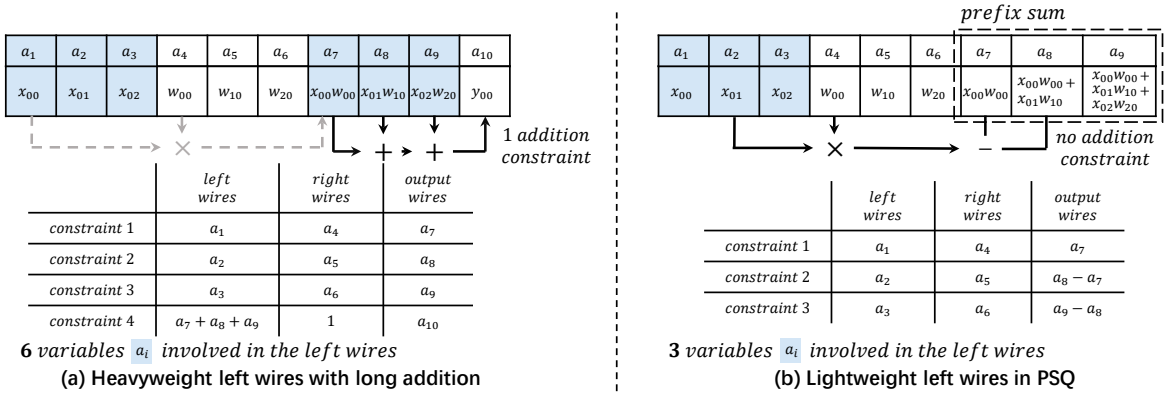


Fig. 5: Comparison: (a) traditional long addition with 6 left-wire variables vs. (b) our PSQ using only 3 variables.

the first column of X is $X_0(Z) = Z^0x_{00} + Z^2x_{10} + Z^4x_{20}$. The matrix Y is converted accordingly. We have:

$$\begin{aligned}(Z^0y_{00} + Z^1y_{01} + Z^2y_{10} + Z^3y_{11} + Z^4y_{20} + Z^5y_{21}) \\ = (Z^0x_{00} + Z^2x_{10} + Z^4x_{20}) \cdot (Z^0w_{00} + Z^1w_{01}) \\ + (Z^0x_{01} + Z^2x_{11} + Z^4x_{21}) \cdot (Z^0w_{10} + Z^1w_{11})\end{aligned}$$

where only 2 multiplications are needed. We generalize CRPC's transformation for matrix multiplication $Y^{a\times b} = X^{a\times n} \times W^{n\times b}$ as:

$$\sum_{j=0}^{b-1}\sum_{i=0}^{a-1} Z^{ib+j}y_{ij} = \sum_{k=0}^{n-1}\left(\sum_{i=0}^{a-1} Z^{ib}x_{ik}\right)\left(\sum_{j=0}^{b-1} Z^{j}w_{kj}\right)$$

where only n constraints are needed, where as $a\cdot b\cdot n$ constraints are needed in vanilla circuits.

Polynomial multiplication offers significant advantages in computational efficiency, especially when representing matrix multiplication in ZKP protocols. Our proposed CRPC reduces the constraint complexity from $O(n^3)$ to $O(n)$ for matrix multiplication. The proving efficiency of zk-SNARKs systems is thereby improved by a significant margin. The overall proving time of ZKP-based matrix multiplication of different sizes in Transformer layers can be $7 \sim 9\times$ faster.

B. Prefix-Sum Query (PSQ)

The CRPC method notably decreases the constraint count needed for matrix multiplication representation. Additionally, it is observed that the number of left wires influences proving performance. We propose PSQ to reduce the number of left wires. Our insight is that, although an arbitrary number of additions can be encapsulated within a single constraint, a prolonged sequence of additions can result in a considerable computational overhead, leading to a large number of left wires number, as is shown in Figure 5(a). Consider a dot product in matrix multiplication, represented as $y_{00} = x_{00}\cdot w_{00} + x_{01}\cdot w_{10} + x_{02}\cdot w_{20}$. This calculation requires 4 constraints. The initial three constraints calculate the intermediate products: $x_{00}w_{00}$, $x_{01}w_{10}$, and $x_{02}w_{20}$, which are assigned to variables a_7, a_8, and a_9. To achieve the final result y_{00}, one more addition is required, incorporating the three intermediate variables a_7, a_8, and a_9 in the left wires. Consequently, this approach uses 6 left variables/wires. Heavy left wires in large matrices can be computationally demanding. Then, we introduce PSQ to avoid extended additions in matrix multiplication, as illustrated in 5 (b). Instead of holding actual intermediate product values in a_7, a_8, and a_9, we record their prefix sums. Specifically, $a_7 = x_{00}w_{00}$, $a_8 = a_7 + x_{01}w_{10}$, $a_9 = a_8 + x_{02}w_{20}$.

Here, the final result y_{00} is directly available in a_9, eliminating the need for an additional constraint for the long addition. PSQ reduces the left wire variables to only 3, making it a more efficient approach for both verification and proving stages.

The proposed PSQ effectively reduces the complexity. Consider the general matrix multiplication $Y^{a\times b} = X^{a\times n} \times W^{n\times b}$. There are $(a\cdot n + a\cdot b\cdot n)$ variables involved, where the $(a\cdot b\cdot n)$ intermediate products make proving rather complex. With PSQ, the proving is only associated with the $(a\cdot n)$ variables, and the complexity is reduced from $O(n^3)$ to $O(n^2)$. PSQ effectively contributes to a lightweight R1CS, and the computation of the R1CS thus becomes significantly more efficient. Specifically, the cost of computing the R1CS is reduced by approximately 70% during the proving phase. By building on the foundation of CRPC, PSQ further reduces the proving cost for matrix multiplication by 30%. This results in a total speedup of 12× for the proving time.

C. Nonlinear-Function Approximation

CRPC and PSQ significantly improves the efficiency of proving matrix multiplication with ZKP. To demonstrate the efficiency of our zkVC, we apply it to verifiable Transformer inference. To verify the correctness of computation in transformers, we need to verify the non-linear functions like $SoftMax$, and GeLU with ZKP. However, ZKP can not directly support these non-arithmetic functions. We design accurate arithmetic approximations for these complex non-arithmetic functions. Given a vector $x \in \mathbb{R}^d$, the $SoftMax$ function is defined as $SoftMax_i(x) = e^{x_i}/\sum_{j\in[d]} e^{x_j}$. The main challenge is to accurately express the exponential function in ZKP constraints. Although the exponential function can not be directly represented by addition and multiplication, it is possible to closely approximate the exponential function on negative inputs. Based on this idea, we show how the SoftMax function is verified in our design as follows.

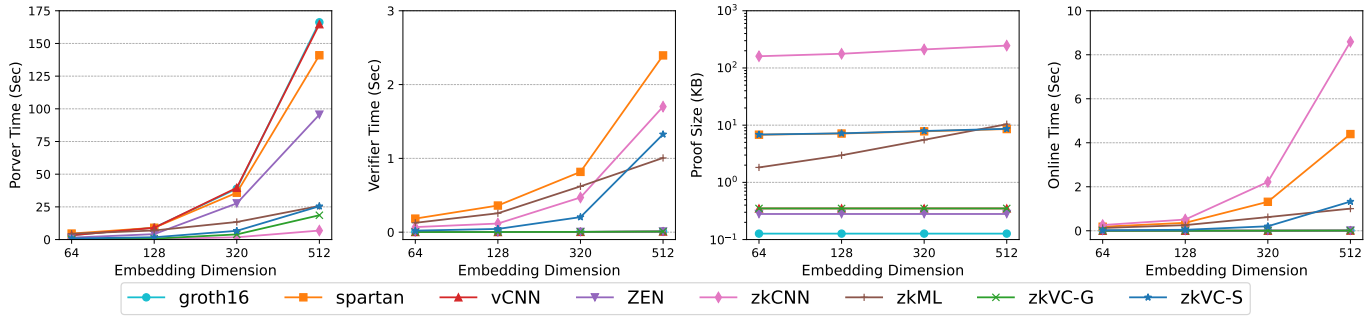


Fig. 6: A comparison of zkVC and prior works on matrix multiplication. zkVC leads in proving time of all non-interactive methods, close to interactive zkCNN, and excels in verification time, proof size, as well as online verification duration.

Computing x_{max}. We first normalize the input vector x by $(x - x_{max})$, where x_{max} is the max element in x. The subtraction can easily be encoded in ZKP circuits. To verify the max computation, we check two constraints: (1) $x_{max} \geq x_j$ for all $j \in [d]$ and (2) $\prod_j (x_{max} - x_j) = 0$. The first constraint ensures x_{max} is greater than all other values in x and the second constraint ensures x_{max} is indeed one of the values from x. Since ZKP supports comparison operations by bit-decomposition [7], these constraints are compatible with ZKP.

Approximating e^x on negative inputs. After normalizing x by $(x - x_{max})$, all elements in the resulting vector are negative. It is evident that computing $SoftMax(x - x_{max})$ is equivalent to computing $SoftMax(x)$. Thus, the SoftMax function can be computed as $SoftMax_i(x) = e^{x_i - x_{max}} / \sum_{j \in [d]} e^{x_j - x_{max}}$, where only the exponential function on negative inputs is needed to compute. We approximate e^x on negative inputs using the Taylor series:

$$e^x \approx \begin{cases} 0, & \text{if } x < T \\ (1 + x/2^n)^{2^n}, & \text{if } x \in [T, 0]. \end{cases}$$

where T is the pre-defined threshold deciding the clipping branch. We apply a two-bit decomposition to compare x with T and compute the division by 2^n. The power-to-2^n computation is essentially a series of multiplications that can be easily encoded in ZKP circuits.

Putting together, we verify the SoftMax function in ZKP via a close approximation using three sets of bit decomposition and two sets of multiplication. The GELU activation function is used in NLP transformers such as BERT. The GELU function is defined as $\mathsf{GELU}(x) = 0.5x\left(1 + \mathsf{Tanh}[\sqrt{2/\pi}(x + 0.044715x^3)]\right)$. Similar to the exponential function in $SoftMax$, the Tanh function can not be directly represented by addition and multiplication gates. We use polynomial approximation to represent the GELU function efficiently. Specifically, we use $\mathsf{GELU}(x) \approx x^2/8 + x/4 + 1/2$.

IV. Experimental Methodology

Models and Datasets. In our experiments, we explored both computer vision and language transformers. For computer vision, we tested three vision transformer architectures on different datasets, adapting settings from [32], [33] for small datasets. On CIFAR-10, the ViT configuration was 7 layers, 4 heads, a hidden dimension of 256, and a patch size of 4. For Tiny-ImageNet, we used 9 layers, 12 heads, a hidden dimension of 192, and a patch size of 4. On ImageNet, a hierarchical architecture from [34], [35] was implemented with 12 layers and 4 stages, with embedding dimensions of 64, 128, 320, 512. For NLP, we chose a BERT model with 4 layers, 4 heads, and an embedding dimension of 256. This model was fine-tuned and assessed on GLUE benchmarks [36], including MNLI, QNLI, SST-2, and MRPC tasks.

Implementation Details. In our experiments, we utilized groth16 from libsnark [37] and Spartan [13] for the ZKP backend, referred to as zkVC-G and zkVC-S, respectively. These cryptographic tests were conducted on AMD Ryzen Threadripper PRO 3955WX 16-Core CPU systems with 128GB RAM, running Ubuntu 22.04.1. Transformer model experiments were performed on NVIDIA GeForce RTX 3090 GPUs. For Transformer architectures, zkVC was built on ViT [4] and MetaFormer's frameworks [35], incorporating efficient token mixers like scaling attention modules [38], [39], linear transformation modules [40], average pooling, and differentiable NAS [41]. We employed the quantization technique from [42] for converting model parameters to integer formats. This marks the first instance of verifiable Transformers being tested on the ImageNet dataset [5].

V. Results

A. Micro-benchmarks

Matrix Multiplication Benchmark. Figure 6 shows a comparison of zkVC with previous works on matrix multiplication benchmarks. The dimensions of the matrices are set according to the embedding layers in ViT [4], i.e., $[\#tokens, dim_1] \times [dim_1, dim_2]$. The number of tokens are set to 49 for simplicity. For example, when the embedding dimension is 128, the dimension of matrix multiplication is $[49, 64] \times [64, 128]$. For non-interactive schemes, zkVC-G is based on groth16 [2], and zkVC-S on Spartan [13]. vCNN [6] and ZEN [17] also use groth16, while Kang's [8] employs halo2 [43]. The interactive scheme included is zkCNN [7], with vanilla groth16 and Spartan serving as baselines. zkVC significantly enhances proving time, achieving 5 to 12 times faster performance than the groth16 and Spartan baselines. Among non-interactive options, zkVC-G stands out for its proving efficiency. Though

TABLE II: Ablation study on matrix multiplication microbenchmark.

CRPC	PSQ	groth16		Spartan	
		Prove(s)	Verify(s)	Prove(s)	Verify(s)
X	X	9.12	0.002	9.04	0.36
X	✓	8.69	0.002	8.95	0.32
✓	X	1.01	0.002	1.79	0.08
✓	✓	0.73	0.002	1.75	0.05

TABLE III: Comparison of various token mixers with our zkVC on ViT Models. SoftApprox. includes approximated SoftMax, Softfree-S (scaling), Softfree-P (pooling), with $\mathcal{P}_G$ for Groth16 and $\mathcal{P}_S$ for Spartan.

Dataset	Model	Top1(%)	$\mathcal{P}_G$(s)	$\mathcal{P}_S$(s)
Cifar-10	SoftApprox.	93.5	725.2	1006.2
	SoftFree-S	88.3	568.4	742.8
	SoftFree-P	75.1	262.7	300.6
	zkVC	91.6	458.6	591
Tiny ImageNet	SoftApprox.	60.5	1609.6	2197.4
	SoftFree-S	51.4	1004.9	1348.8
	SoftFree-P	42.7	443.7	503.6
	zkVC	55.8	879.3	1161.4
ImageNet	SoftApprox.	81	10700	12857.7
	SoftFree-S	78.5	4521.3	5812.7
	SoftFree-P	77.2	2904	3667.8
	zkVC	80.3	3457.1	4417.1

zkCNN is about twice as fast in proving compared to zkVC-G, it requires interaction and suffers from slower verification and larger proof sizes. Specifically, zkCNN's verification is up to 200 times slower and its proofs are 1 to 2 orders of magnitude larger than those of zkVC. The online time means the time that the client and server need to maintain online during the proving. zkCNN also demands additional verifier online time due to its interactive nature.

Technical Ablation Study. In Table II, we present the latency results from a matrix multiplication microbenchmark to highlight the effectiveness of the proposed CRPC and PSQ in transformer patch embedding layers. CRPC significantly reduces proving time, boosting the groth16 backend by approximately 9× and the Spartan backend by around 5×. While groth16 maintains a consistent verification time, CRPC cuts Spartan's verification time by about 4×. The application of PSQ further speeds up groth16, achieving up to a 12× faster rate. Directly applying PSQ to Groth16 can also improve proving time, but the enhancement is more significant when PSQ is coupled with CRCP. However, PSQ's impact on Spartan's proving time is minimal, this is because CRPC already reduces the number of constraints for both backends, while PSQ primarily simplifies specific complex queries for groth16.

B. End-to-end Performance

Vision. Table III illustrates zkVC's accuracy-latency balance on three popular vision datasets. For smaller datasets like Cifar-10 and Tiny ImageNet, using average pooling instead of SoftMax self-attention notably reduces accuracy. Although scaling attention models are more accurate, their efficiency gain in ZKP proving is minimal due to low-resolution images (e.g., 32×32 in Cifar-10) and fewer transformer input tokens (e.g., 64 tokens for 4-size patches). Average pooling alone struggles with low-res images, and scaling attention alone shows limited efficiency for short token sequences. zkVC, blending $SoftMax$ self-attention with $SoftMax$-free options, achieves around 40% faster proving on Cifar-10 with under 2% accuracy loss, and about 50% faster on Tiny ImageNet with less than 5% accuracy loss.

TABLE IV: Comparison of various token mixers with our zkVC on NLP Models. SoftFree-L denotes a model using linear transformation for token mixing.

Model	Acc. on Tasks(%)				$\mathcal{P}_G$(s)	$\mathcal{P}_S$(s)
	MNLI	QNLI	SST-2	MRPC		
SoftApprox.	74.5	83.9	85.8	71.2	1299.5	1793.3
SoftFree-S	72.7	81.1	85.2	70.4	917.1	1201.4
SoftFree-L	67.3	75.3	84.5	68.7	680.8	782.0
zkVC	70.8	80.2	84.7	69.3	798.9	992.2

High-resolution ImageNet images, with many input tokens (e.g., 3136 for a 224×224 image with patch size 4), strain SoftMax self-attention's quadratic complexity. Scaling attention's linear complexity better manages these long sequences. SoftMax-free methods, like scaling attention and average pooling, excel on larger datasets, speeding up computations by 60% to 70% with less than 5% accuracy loss. zkVC achieves similar speedups with under 1% accuracy loss, likely due to reintegrating SoftMax self-attention in later transformer layers with shorter token sequences.

NLP. Table IV shows zkVC's evaluation on NLP transformers like BERT. Linear transformation improves efficiency by 50% but can drop accuracy by up to 7% on MNLI. Scaling attention increases proving efficiency by 30%, with better, more stable performance across tasks. zkVC is about 15% faster than scaling attention models while being around 3% more accurate on average compared to linear transformation. The results indicate that replacing all *SoftMax* attention with SoftMax-free alternatives isn't always ideal for accuracy and latency demands. zkVC, using our planner, combines a hybrid transformer architecture. It matches *SoftMax*-centric models in accuracy and outperforms Linear Attention models in latency.

VI. Conclusion

In this paper, we introduced *zkVC*, an efficient zk-SNARK construction designed to optimize matrix multiplication verification. Traditional ZKP approaches for verifying matrix multiplication often require an excessive number of constraints, leading to high computational overhead. *zkVC* addresses this challenge by leveraging CPRC to minimize constraints and PSQ to reduce variables, achieving a 12× improvement in proof efficiency over prior methods. We further demonstrated *zkVC*'s effectiveness in verifiable Transformer inference, verifying the integrity of ViT models efficiently. Given that matrix multiplication underpins a wide range of applications, we believe *zkVC* offers a significant step forward in enhancing the computational integrity of these applications.

References

[1] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2016.

[2] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 2016, pp. 305–326.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[6] S. Lee, H. Ko, J. Kim, and H. Oh, "vcnn: Verifiable convolutional neural network based on zk-snarks," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[7] T. Liu, X. Xie, and Y. Zhang, "Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2968–2985.

[8] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, "Scaling up trustless dnn inference with zero-knowledge proofs," 2022.

[9] J. Weng, J. Weng, G. Tang, A. Yang, M. Li, and J.-N. Liu, "pvcnn: Privacy-preserving and verifiable convolutional neural network testing," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2218–2233, 2023.

[10] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2111–2128.

[11] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. Springer, 2020, pp. 738–768.

[12] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 926–943.

[13] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *Annual International Cryptology Conference*. Springer, 2020, pp. 704–737.

[14] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[15] J. Keuffer, R. Molva, and H. Chabanne, "Efficient proof composition for verifiable computation," in *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. López, J. Zhou, and M. Soriano, Eds., vol. 11098. Springer, 2018, pp. 152–171.

[16] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, "Veriml: Enabling integrity assurances and fair payments for machine learning as a service," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 10, pp. 2524–2540, 2021.

[17] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, "Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences," Cryptology ePrint Archive, Paper 2021/087, 2021.

[18] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive oracle proofs," in *Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II 14*. Springer, 2016, pp. 31–60.

[19] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent snarks from dark compilers," in *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. Springer, 2020, pp. 677–706.

[20] Q. Lou and L. Jiang, "She: A fast and accurate deep neural network for encrypted data," *Advances in neural information processing systems*, vol. 32, 2019.

[21] M. Zheng, Q. Lou, and L. Jiang, "Primer: Fast private transformer inference on encrypted data," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[22] Q. Lou, Y. Shen, H. Jin, and L. Jiang, "Safenet: A secure, accurate and fast neural network inference," in *International Conference on Learning Representations*, 2021.

[23] Q. Lou and L. Jiang, "Hemet: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture," in *International conference on machine learning*. PMLR, 2021, pp. 7102–7110.

[24] Q. Lou, M. Santriaji, A. W. B. Yudha, J. Xue, and Y. Solihin, "vfhe: Verifiable fully homomorphic encryption with blind hash," *arXiv preprint arXiv:2303.08886*, 2023.

[25] M. H. Santriaji, J. Xue, Y. Zhang, Q. Lou, and Y. Solihin, "Dataseal: Ensuring the verifiability of private computation on encrypted data," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 78–78.

[26] M. Kumar, J. Xue, M. Zheng, and Q. Lou, "Tfhe-coder: Evaluating llm-agentic fully homomorphic encryption code generation," *arXiv preprint arXiv:2503.12217*, 2025.

[27] Y. Zhang, M. Zheng, Y. Shang, X. Chen, and Q. Lou, "Heprune: Fast private training of deep neural networks with encrypted data pruning," in *Advances in Neural Information Processing Systems*, vol. 37. Curran Associates, Inc., 2024, pp. 51 063–51 084.

[28] Y. Zhang, J. Xue, M. Zheng, M. Xie, M. Zhang, L. Jiang, and Q. Lou, "Cipherprune: Efficient and scalable private transformer inference," *arXiv preprint arXiv:2502.16782*, 2025.

[29] X. Deng, S. Fan, Z. Hu, Z. Tian, Z. Yang, J. Yu, D. Cao, D. Meng, R. Hou, M. Li *et al.*, "Trinity: A general purpose fhe accelerator," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 338–351.

[30] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 926–943.

[31] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.

[32] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi, "Escaping the big data paradigm with compact transformers," *arXiv preprint arXiv:2104.05704*, 2021.

[33] W. Zeng, M. Li, W. Xiong, W. Lu, J. Tan, R. Wang, and R. Huang, "Mpcvit: Searching for mpc-friendly vision transformer with heterogeneous attention," *arXiv preprint arXiv:2211.13955*, 2022.

[34] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[35] W. Yu, M. Luo, P. Zhou, C. Si, Y. Zhou, X. Wang, J. Feng, and S. Yan, "Metaformer is actually what you need for vision," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 819–10 829.

[36] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[37] "libsnark," https://github.com/scipr-lab/libsnark.

[38] Z. Shen, M. Zhang, H. Zhao, S. Yi, and H. Li, "Efficient attention: Attention with linear complexities," 2020.

[39] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803.

[40] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon, "Fnet: Mixing tokens with fourier transforms," *arXiv preprint arXiv:2105.03824*, 2021.

[41] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[42] M. Wang, S. Rasoulinezhad, P. H. W. Leong, and H. K.-H. So, "NITI: Training integer neural networks using integer-only arithmetic," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 3249–3261, nov 2022.

[43] S. Bowe, J. Grigg, and D. Hopwood, "Recursive proof composition without a trusted setup," *Cryptology ePrint Archive*, 2019.

MMDFL: Multi-Model-based Decentralized Federated Learning for Resource-Constrained AIoT Systems

Dengke Yan[1], Yanxin Yang[1], Ming Hu[2], Xin Fu[3], and Mingsong Chen[1]

[1]MoE Engineering Research Center of SW/HW Co-Design Technology and Application, East China Normal University
[2]School of Computing and Information Systems, Singapore Management University
[3]Department of Electrical & Computer Engineering, University of Houston

***Abstract*—Along with the prosperity of Artificial Intelligence (AI) techniques, more and more Artificial Intelligence of Things (AIoT) applications adopt Federated Learning (FL) to enable collaborative learning without compromising the privacy of devices. Since existing centralized FL methods suffer from the problems of single-point-of-failure and communication bottleneck caused by the parameter server, we are witnessing an increasing use of Decentralized Federated Learning (DFL), which is based on Peer-to-Peer (P2P) communication without using a global model. However, DFL still faces three major challenges, i.e., limited computing power and network bandwidth of resource-constrained devices, non-Independent and Identically Distributed (non-IID) device data, and all-neighbor-dependent knowledge aggregation operations, all of which greatly suppress the learning potential of existing DFL methods. To address these problems, this paper presents an efficient DFL framework named MMDFL based on our proposed multi-model-based learning and knowledge aggregation mechanism. Specifically, MMDFL adopts multiple traveler models, which perform local training individually along their traversed devices, accelerating and maximizing knowledge learning and sharing among devices. Moreover, based on our proposed device selection strategy, MMDFL enables each traveler to adaptively explore its next best neighboring device to further enhance the DFL training performance, taking into account issues of data heterogeneity, limited resources and catastrophic forgetting phenomenon. Experimental results from simulation and a real testbed show that, compared with state-of-the-art DFL methods, MMDFL can not only significantly reduce the communication overhead but also achieve better overall classification performance for both IID and non-IID scenarios.**

***Index Terms*—AIoT, decentralized federated learning, multi-model learning, resource-constrained, stochastic gradient descent.**

I. Introduction

The increasing popularity of Deep Learning (DL) accelerates the shift from the Internet of Things (IoT) to Artificial Intelligence of Things (AIoT) [1]–[4], where devices are equipped with DL models to perform precise detection and intelligent control [5]. Although these DL models play an important role in various critical domains (e.g. object detection [6], intelligent transportation [7], healthcare [8], and EDA [9]), due to the limited computing resources of the devices and restricted access to global data, their training and inference performance are difficult to guarantee, especially when dealing with numerous heterogeneous AIoT devices with different computing power and communication capabilities [10]. To quickly figure out both the training and inference results of devices, more and more AIoT systems resort to cloud computing [11], which provides tremendous computational capabilities and flexible device management strategies. However, such a cloud-based architecture cannot essentially enhance the inference accuracy of AIoT devices, since privacy concerns prohibit them from sharing their local data with each other. Consequently, both the training and the inference capabilities of local models are greatly suppressed.

As a promising distributed machine learning paradigm, Federated Learning (FL) [12] enables efficient collaborative model training among AIoT devices [13], [14]. Instead of sharing local sensitive data directly among devices, FL only exchanges device models or their gradients, thus enhancing the inference capability of AIoT devices without compromising their data privacy. Although FL is good at knowledge sharing among devices, most existing FL methods are built on top of a cloud-client architecture, which struggles with large-scale deployment [3], [15]. This is mainly because, within the cloud-client architecture, the cloud-based central server can easily result in the single-point-of-failure, where adversaries can easily break the whole system's privacy by attacking the central server. Moreover, to perform the Federated Averaging (FedAvg) operation, existing FL methods force a large set of selected clients to communicate with the central server, imposing a massive burden of communication on the central server [16]. Things get even worse when some devices have limited resources (e.g., computing power and network bandwidth), greatly prolonging overall training time [17].

Instead of relying on a central server, Decentralized Federated Learning (DFL) [18]–[20] requires all devices to communicate only with their neighboring devices for knowledge aggregation. In this way, DFL can not only address the communication bottleneck caused by the cloud server, but also improve the resilience of device privacy and the robustness against clients' failures. Despite all these merits, since all devices are required to participate in each training round synchronously, existing DFL methods suffer from various problems during their local training processes. Like traditional FL methods, DFL is not friendly for AIoT applications with resource-constrained devices [21]. For example, when DFL training involves devices with limited computing power or network bandwidth, according to the "wooden barrel theory", the training time for each round is determined by the weakest devices, thus prolonging the overall training process. Moreover, due to the phenomenon of *weight divergence*, the classification performance of existing DFL methods degrades significantly when handling scenarios with non-Independent and Identically Distributed (non-IID) on-device data [22]. Clearly, *how to develop a DFL method to quickly achieve higher classification accuracy for resource-constrained devices within non-IID scenarios is becoming a major bottleneck in the design of modern AIoT applications*.

Although DFL is based on Peer-to-Peer (P2P) communication between devices and their neighbors, it still relies on FedAvg to implement its knowledge aggregation operations. Since each DFL training round involves all devices, the entanglements of interconnected devices strongly suppress the learning potential of device models in DFL. In this paper, we present a novel multi-model-based DFL training scheme named MMDFL, which breaks through the FedAvg limit to enable more efficient training of local models by decoupling the multi-to-one relation imposed by FedAvg. Unlike traditional DFL methods, MMDFL maintains a set of traveler nodes, where each traveler model traverses a series of neighboring devices individually and performs local model training hop-by-hop. During DFL training, if a device has $n>1$ travelers simultaneously, MMDFL will first perform the FedAvg operation on these travelers to update

its local model and then use it for local training. After that, the device will forward the trained local model to its n best candidate neighbors as new travelers. Note that device selection plays an important role in MMDFL. An unwise selection of neighboring devices may easily result in: i) the *straggler problem*, where fresh knowledge learned by travelers can be easily ruined by stale knowledge learned by the least frequently updated devices (i.e., stragglers) [23]; or ii) the *catastrophic forgetting problem*, where travelers may quickly forget the knowledge learned on their previously traversed devices.

To address these issues, MMDFL deliberately selects the best successor neighboring device for each traveler, considering various factors caused by the limited device resources. Compared with existing DFL methods, since traveler models can access more device data along their routes with less skewness for local training, MMDFL can achieve higher classification accuracy and faster training convergence. This paper makes the following three major contributions:

- We introduce a novel *multi-model-based DFL training scheme* using our proposed travel models rather than traditional federated averaging, which improves and accelerates knowledge sharing between devices with much less communication overhead.
- We develop an *adaptive traveler model-oriented device selection strategy*, which maximizes the knowledge learning and sharing capabilities of traveler models under the constraints imposed by underlying non-IID device data, and limited network bandwidth and computing power of devices.
- We conduct extensive experiments using both simulation and a real testbed to demonstrate the superiority of MMDFL in terms of training performance and overall communication overhead.

II. Related Work

As an emerging distributed collaborative learning paradigm, various Decentralized Federated Learning (DFL) methods [18] have been investigated to address the problems of single-point-of-failure and communication bottleneck, which are common in centralized FL caused by the parameter server. However, due to resource-constrained devices and non-IID data distributions, existing DFL methods [18]–[20] suffer greatly from the problem of low classification accuracy and slow convergence.

To improve the training performance of DFL, especially within non-IID scenarios, various model training and aggregation optimization methods have been proposed. For example, DFedSAM [24] leverages gradient perturbation to generate local flat models via Sharpness Aware Minimization, which searches for models with uniformly low loss values. By searching for models with uniformly low loss values, DFedSAM can significantly improve the accuracy of aggregated models. In [25], SGP adopts a new model aggregation method called PUSHSUM, which can be used together with stochastic gradient updates to effectively improve model training performance. In addition to these methods, how to mitigate the negative impacts of data heterogeneity is becoming a promising alternative to improve DFL training. For example, DeSA [26] aligns output features between clients by synthesizing a global anchor, facilitating knowledge transfer between them. For each client, FedHP [27] dynamically selects different neighbors with the closest consensus distances for knowledge aggregation in each round of communication, thus effectively alleviating the weight divergence problem.

Since the knowledge aggregation operation of existing DFL methods is still based on FedAvg, when dealing with numerous resource-constrained devices, their communication overhead is not negligible. Therefore, various methods are introduced to save communication resources by transmitting partial models instead of their full versions. For example, YOGA [28] ranks the importance of each layer of a model according to the amount of parameter variation, and transmits only the most important model layers to its neighboring clients. In [29], DFedPGP divides a model into two parts (i.e., the shared model part and the private model part) and conducts training on them as a whole, but only transmits the shared model parts between clients, thus saving the communication bandwidth.

Although the above solutions are promising for optimizing the DFL training performance, most of them are based on the FedAvg-like knowledge aggregation paradigm, neglecting the conflicting and time-consuming interactions between neighboring devices. To the best of our knowledge, MMDFL is the first attempt to break the FedAvg limit to enable more efficient training of local models. By resorting to our proposed multi-model-based knowledge training and sharing paradigm, MMDFL can not only significantly reduce the communication overhead, but also achieve better classification accuracy, thus inherently improving the overall DFL training performance.

III. Our Approach

A. Notations for DFL

The goal of this paper is to achieve the best possible overall classification performance for all devices within an AIoT system. Like existing DFL methods [30]–[32], we use an undirected graph $G(V, E)$ to represent the device communication topology for DFL, where V denotes the set of K devices and E denotes the set of communication channels between devices. In our approach, each device $v_i \in V$ is modeled as a triple, containing local raw data d_i, local training time τ_i, and current update status s_i of v_i. Here, s_i indicates the index of the last communication round in which v_i is trained. When there is a communication channel between devices v_i and v_j, we denote it as an edge $e \in E$ in the form of (i, j, b_{ij}), where $i<j$ and b_{ij} represents the communication bandwidth from v_i to v_j. Note that $b_{ij}=b_{ji}$. We use $\bigcup_{j \in N(i)}\{v_j\}$ to represent the set of neighboring devices of v_i, where $N(i) = \{j|(i, j, b_{ij}) \in E\}$.

In the t^{th} communication round, each $v_i \in V$ updates its local model w_i based on its local raw data d_i as follows:

$$\hat{\theta_i^t} = \theta_i^t - \mu \nabla \mathcal{L}(w_i(x_i; \theta_i^t), y_i), \tag{1}$$

where x_i and y_i represent the samples and labels in d_i, respectively, θ_i^t is the model parameters, μ is the learning rate of the model training, and $\mathcal{L}$ is the cross-entropy loss function. After training, device v_i sends its updated model parameters $\hat{\theta_i^t}$ to its neighboring devices. Meanwhile, it receives the model parameters from its neighboring devices and aggregates these model parameters according to the following equation:

$$\theta_i^{t+1} = \frac{\hat{\theta_i^t} + \sum_{j \in N(i)} \hat{\theta_j^t}}{|N(i)| + 1}, \tag{2}$$

where $|N(i)|$ is the number of neighboring devices of v_i. The goal of the DFL training process is to optimize the model parameters to find a θ^* that minimizes the training losses of each device, which can be formulated as:

$$\theta^* = \arg\min_{\theta} \frac{1}{K} \sum_{i=1}^{K} \frac{1}{|d_i|} \mathcal{L}(w(x_i; \theta), y_i). \tag{3}$$

B. Overview of MMDFL

From Equation 2, we can find that the knowledge aggregation operation of traditional DFL methods is still in a FedAvg-like style involving all neighboring device models. Due to the mismatch of the data distributions and computing power of neighboring devices, the

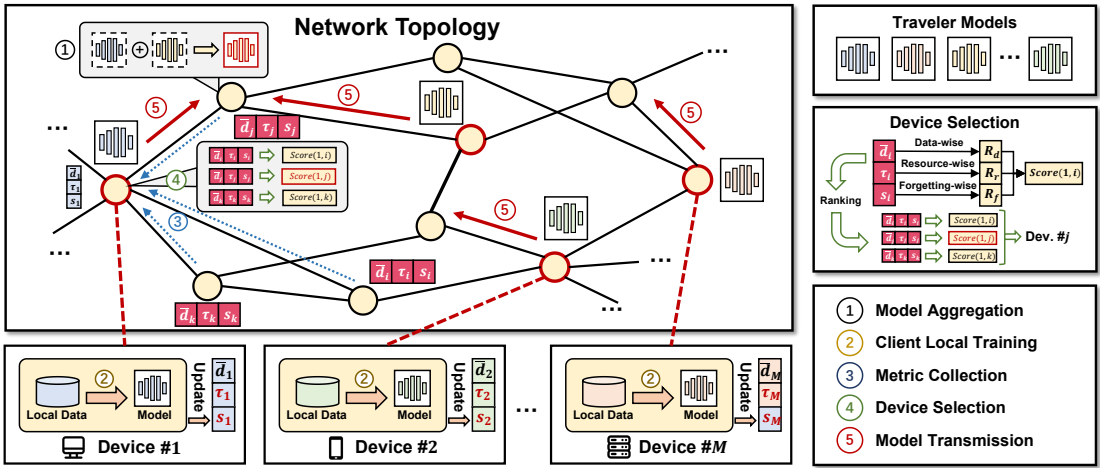


Fig. 1: Framework and workflow of MMDFL.

entanglements of interconnected devices will strongly suppress the learning potential of the final device models. As an alternative, Figure 1 presents the framework and workflow of our MMDFL method based on a set of traveler models, where each traveler consecutively traverses neighboring devices and conducts local training hop-by-hop. Assume that there are K devices and M traveler models in total, where typically M is much smaller than K. In each communication round, a device is called *activated* if it hosts at least one traveler. After local training, the updated traveler models will be forwarded to the best neighbors of the activated device using our *adaptive device selection strategy* considering the constraints imposed by non-IID device data, limited network bandwidth, and the computing power of devices. Specifically, each MMDFL communication round involves five consecutive key steps as follows:

Step 1: Model Aggregation. When a device receives n>1 traveler models in the last communication round, MMDFL will apply FedAvg on these models to obtain a new aggregated model. If the device receives only one traveler model (i.e., n=1), no aggregation is necessary, since such a traveler model is the aggregated model.

Step 2: Client Local Training. Each activated device trains its aggregated model using local raw data, respectively. After training, MMDFL updates all the n traveler models using the trained model.

Step 3: Metric Collection. Each activated device collects the metrics of all its neighboring devices, including their data distributions, training time, and update status.

Step 4: Device Selection. Based on the collected device metrics, each activated device calculates a selection score for each neighboring device based on our adaptive device selection strategy. MMDFL selects the best n neighbors with the smallest scores.

Step 5: Model Transmission. Each activated device sends its n updated traveler models to its n best selected neighbors.

C. Adaptive Traveler Model-Oriented Device Selection Strategy

During traversal, due to constrained device resources and imbalanced on-device data, an improper selection of the next neighboring device made by a traveler will easily result in the notorious straggler problem or the catastrophic forgetting problem, degrading the overall training performance. To avoid such problems, we propose an adaptive traveler model-oriented device selection strategy to guide device traversal by traveler models. When a specific traveler model w is trained on a device v_i, it collects the metrics from both v_i and all its neighboring devices $\bigcup_{j\in N(i)}\{v_j\}$. Specifically, for a device v_i, the metrics include the distribution of local data d_i (i.e., $\overline{d_i}$), the local training time τ_i, and the current update status s_i. After collecting the metrics, MMDFL will analyze these metrics from data-wise, resource-wise, and forgetting-wise aspects.

Data-wise Aspect. To mitigate the negative impacts of data heterogeneity, for the overall training data used by each traveler model, MMDFL attempts to make its distribution as close to a uniform distribution as possible. Specifically, the traveler model w possesses a knowledge vector l_{v_i}, representing the distribution of all the data used to train w_{v_i}. The length of l_{v_i} is the number of classification categories c. After training the traveler model w_{v_i} using local raw data d_i, the knowledge vector will be updated as $l_{v_i} = l_{v_i} + \overline{d_i}$. When an activated device v_i receives the data distribution $\overline{d_j}$ from a neighboring device v_j, the distribution distance between l_{v_i} and $\overline{d_j}$ can be calculated as follows:

$$Dis(l_{v_i}, \overline{d_j}) = \sum_{p=1}^{c}\left(\frac{l_{v_i}[p] + \overline{d_j}[p]}{\sum_{q=1}^{c}(l_{v_i}[q] + \overline{d_j}[q])} - \frac{1}{c}\right)^2, \tag{4}$$

where the notion "$x[y]$" indicates the y^{th} item of vector x. After calculating the distribution distances between a device and all its neighboring devices, the data-wise score of these neighboring devices can be calculated as:

$$R_d(i,j) = \frac{Dis(l_{v_i}, \overline{d_j})}{\sum_{k\in N(i)} Dis(l_{v_i}, \overline{d_k})}. \tag{5}$$

Resource-wise Aspect. To achieve adequate training of a traveler model in AIoT scenarios with various resource-constrained devices, MMDFL prioritizes neighboring devices with larger communication bandwidth and computing power. Therefore, we calculate the resource-wise score of each neighboring device as:

$$R_r(i,j) = \frac{|\theta|/b_{ij} + \tau_j}{\sum_{k\in N(i)}(|\theta|/b_{ik} + \tau_k)}. \tag{6}$$

Forgetting-wise Aspect. To prevent a traveler model from forgetting what it has learned in the past, MMDFL prioritizes selecting devices that have been least recently visited by traveler models, where the forgetting-wise score of each neighboring device is calculated as:

$$R_f(i,j,t) = \frac{(t - s_j + 1)^{-a}}{\sum_{k\in N(i)}(t - s_k + 1)^{-a}}, \tag{7}$$

where t is the index of the current communication round, and a is a constant to control the degree of prioritization for devices that have not been accessed for a long time.

For a pair of neighboring devices v_i and v_j, the overall selection score can be calculated as:

$$Score(i,j,t) = \eta_d \cdot R_d(i,j) + \eta_r \cdot R_r(i,j) + \eta_f \cdot R_f(i,j,t), \quad (8)$$

where η_d, η_s, and η_f are coefficients to balance the three aspects satisfying $\eta_d + \eta_s + \eta_f = 1$. To maximize knowledge learning and sharing among devices, each traveler selects the neighboring device with the lowest score as its next hop for local training.

D. Implementation of MMDFL

Algorithm 1 Implementation of MMDFL

Input: i) K, the number of devices; ii) $G = \{V, E\}$, network topology of devices; iii) M, the number of traveler models; iv) l^0, initial knowledge vector; v) θ^0, initial parameters of traveler models; vi) T, the maximum number of communication rounds;
Output: W^T, set of trained traveler models;

1: $\hat{V}^1 \leftarrow$ RandomSelect(V, M)
2: **for** each device $v_i \in \hat{V}^0$ in parallel **do**
3: $\quad w^0_{v_i} \leftarrow (l^0, \theta^0)$
4: **end for**
5: **for** $t = 1$ to T **do**
6: $\quad$ **for** each device $v_i \in \hat{V}^t$ in parallel **do**
7: $\quad\quad w^t_{v_i} \leftarrow$ Aggregate$(l_{rece}, \theta^t_{rece})$
8: $\quad\quad n_{vi} \leftarrow |\theta^t_{rece}|$
9: $\quad$ **end for**
10: $\quad \hat{V}^{t+1} \leftarrow \{\}$
11: $\quad$ **for** each device $v_i \in \hat{V}^t$ in parallel **do**
12: $\quad\quad (l_{v_i}, \theta^t_{v_i}) \leftarrow$ LocalTrain$(w^{t-1}_{v_i}, d_i)$
13: $\quad\quad$ **for** each device $v_{j \in N(i)}$ in parallel **do**
14: $\quad\quad\quad$ device v_j sends $(d_j, \tau_j, s_j, b_{ij})$ to device v_i
15: $\quad\quad$ **end for**
16: $\quad\quad \hat{S}^t_i \leftarrow \{\}$
17: $\quad\quad$ **for** each $j \in N(i)$ **do**
18: $\quad\quad\quad S_{ij} \leftarrow Score(i,j,t)$ // Equation 8
19: $\quad\quad\quad \hat{S}^t_i \leftarrow \hat{S}^t_i \cup \{S_{ij}\}$
20: $\quad\quad$ **end for**
21: $\quad\quad v_{next} \leftarrow$ Selection$(\hat{S}^t_i, n_{vi})$
22: $\quad\quad$ device v_i sends $(l_{v_i}, \theta^t_{v_i})$ to each device in v_{next}
23: $\quad\quad \hat{V}^{t+1} \leftarrow \hat{V}^{t+1} \cup v_{next}$
24: $\quad$ **end for**
25: **end for**
26: **return** $W^T \leftarrow \bigcup_{i=1}^{M} \{w^T_i\}$

Algorithm 1 details the implementation of MMDFL. Lines 1-4 describe the initialization of MMDFL. Specifically, MMDFL randomly selects M devices as initial activated devices, which are the start locations of traveler models. Lines 2-4 initialize the knowledge vector and parameters of each traveler model. Lines 5-25 present the MMDFL training process in the communication round t. Lines 6-9 present that each activated device aggregates the received model parameters θ_{rece} and knowledge vectors l_{rece}. The number of received models is denoted as n_{vi}. Line 10 creates an empty set $\hat{V}^{t+1}$ to collect the newly activated devices in the next round. In Line 12, each activated device trains its traveler model using the local raw data, obtaining the updated knowledge vector l_{v_i} and model parameters $\theta^t_{v_i}$. In Lines 13-15, each neighboring device of device v_i sends its data distribution, training time, update status, and communication overhead to v_i. Line 16 creates an empty set $\hat{S}^t_i$ to collect the selection scores. In lines 17-20, each activated device calculates the selection scores for all neighboring devices according to Equation 8. For each activated device, Lines 21-23 indicate that its traveler model is sent to the best n_{vi} neighboring device with the smallest selection scores. Finally, Line 26 returns the set of well-trained traveler models.

IV. Performance Evaluation

To evaluate the effectiveness of our approach in resource-constrained AIoT scenarios, we implemented MMDFL based on PyTorch (version 2.0.1). For a fair comparison, we adopted the same SGD optimizer with a learning rate of 0.01 and a momentum of 0.5 for all the investigated DFL methods. By default, we set the batch size to 128, the number of local epochs to 5, and the number of communication rounds to 500. All experiments were carried out on an Ubuntu workstation equipped with an Intel i9-13900K CPU, 64GB memory, and an NVIDIA RTX 4090 GPU.

A. Experimental Settings

Device and Communication Settings. We conducted our experiments on an AIoT system with 20 randomly-connected devices. The system consists of three types of devices (i.e., weak, middle, and strong) indicating different computing power, where the quality of device communication can be classified into three categories, i.e., low, middle, and high. Table I lists the device and communication settings, where we use training and communication time to indicate the computing power of devices and the network bandwidth between devices, respectively. We assume that the proportions of devices and communication channels with different settings are the same for the AIoT system. To reflect fluctuations in device computing power and network connection quality in practice, Table I assumes that in each communication round both training and communication time follow Gaussian distributions in the form of $\mathcal{N}(\mu, \sigma^2)$.

TABLE I: Device and communication settings.

Type	Training Time (s)	Quality	Communication Time (s)
Weak	$\mathcal{N}(50, 2)$	Low	$\mathcal{N}(10, 0.5)$
Middle	$\mathcal{N}(20, 2)$	Middle	$\mathcal{N}(5, 0.5)$
Strong	$\mathcal{N}(10, 2)$	High	$\mathcal{N}(2, 0.5)$

Dataset and Model Settings. We investigated three datasets (i.e., CIFAR-10 [33], CIFAR-100, Tiny-ImageNet [34]) and three models (i.e., ResNet-8 [35], VGG-16 [36], and MobileNet [37]) in the experiments. For local training data, we considered various IID and non-IID scenarios to evaluate the performance of MMDFL under different data distributions. Specifically, we adopted the Dirichlet distribution denoted by $Dir(\alpha)$ to control data heterogeneity, where a smaller α indicates greater heterogeneity of the local device data. Note that we used the complete test set of each dataset to test the performance of each method.

B. Performance Comparison

We compared MMDFL with five SOTA DFL methods, i.e., DFedAvgM [18], D-PSGD [38], LD-SGD [39], DFedSAM [24], and DFedPGP [29]. In each communication round, for DFedAvgM, D-PSGD, and DFedSAM, each device sends a complete model to all its neighboring devices and receives the models from them. For LD-SGD, each device performs only local training in the first three of every five communication rounds, and trains and aggregates the model in the last two rounds. For DFedPGP, each device only sends the parameter of the feature extraction layers in the model. Following the original settings in [29], we set the part of a model before the fully connected layer as the feature extraction layers in our experiments. For MMDFL, we set the number of traveler models to 4 by default, where η_d, η_s, and η_f are set to 0.7,0.1,0.2, respectively. Note that since there is no server in DFL, in the five baselines, we used the average accuracy of all device models as the evaluation metric, while in MMDFL, we used the average accuracy of all traveler models.

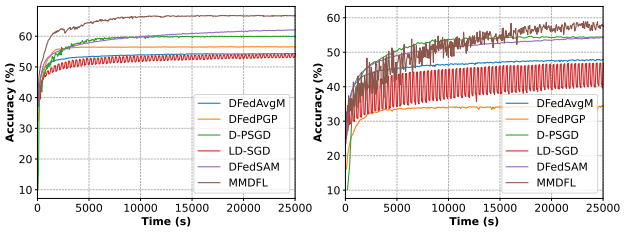
TABLE II: Test accuracy (%) comparison with different datasets and models.

Dataset	Model	Hetero. Settings	DFedAvgM	D-PSGD	LD-SGD	DFedSAM	DFedPGP	MMDFL
CIFAR-10	ResNet-8	IID	54.55±0.03	59.95±0.04	53.71±0.40	63.40±0.03	56.57±0.02	**66.72±0.05**
		$\alpha = 1.0$	56.03±0.23	56.23±0.61	51.81±1.36	61.92±0.07	43.16±0.17	**65.07±0.46**
		$\alpha = 0.5$	49.49±0.62	54.47±0.11	44.01±2.65	56.94±0.08	34.52±0.09	**59.90±0.31**
	VGG-16	IID	75.25±0.03	70.39±0.05	74.74±0.07	79.29±0.06	74.38±0.10	**82.19±0.04**
		$\alpha = 1.0$	74.47±0.67	65.67±2.86	72.42±1.13	78.44±0.12	60.46±0.23	**79.04±0.50**
		$\alpha = 0.5$	70.07±0.46	64.59±0.17	66.12±1.76	72.54±0.14	49.45±0.67	**76.29±0.37**
	MobileNet	IID	57.82±0.08	57.65±0.14	56.34±0.82	65.76±0.04	57.50±0.05	**70.10±0.06**
		$\alpha = 1.0$	54.59±0.12	49.48±0.74	50.27±2.69	61.72±0.11	43.35±0.11	**66.35±0.55**
		$\alpha = 0.5$	45.72±0.22	45.27±0.52	40.15±3.46	55.35±0.08	34.26±0.07	**57.89±1.07**
CIFAR-100	ResNet-8	IID	48.94±0.07	48.99±0.33	41.77±0.65	51.60±0.06	45.43±0.05	**57.21±0.16**
		$\alpha = 1.0$	48.39±0.12	48.35±0.47	39.12±2.94	50.78±0.07	38.40±0.11	**55.66±0.36**
		$\alpha = 0.5$	47.17±0.09	45.76±0.47	37.95±3.80	50.09±0.09	33.40±0.09	**54.18±0.49**
	VGG-16	IID	58.25±0.08	53.42±0.28	53.01±1.27	61.63±0.07	58.57±0.05	**65.07±0.09**
		$\alpha = 1.0$	58.15±0.16	51.33±0.86	52.31±2.38	60.66±0.09	51.67±0.13	**63.44±0.41**
		$\alpha = 0.5$	58.52±0.14	48.88±0.67	51.06±3.39	60.14±0.09	44.56±0.17	**62.12±0.46**
	MobileNet	IID	47.45±0.06	37.38±0.38	38.53±2.55	52.03±0.12	40.60±0.04	**53.84±0.13**
		$\alpha = 1.0$	46.10±0.09	35.48±0.38	35.97±3.81	49.77±0.08	33.22±0.05	**51.79±0.23**
		$\alpha = 0.5$	43.73±0.12	30.94±0.41	33.87±4.63	48.95±0.09	28.49±0.05	**49.50±0.37**
Tiny-ImageNet	ResNet-8	IID	26.42±0.13	25.63±0.24	24.64±0.34	31.13±0.32	20.58±0.15	**35.50±0.38**
		$\alpha = 1.0$	24.88±0.10	24.53±0.38	23.84±0.79	32.01±0.12	19.04±0.16	**33.98±0.34**
		$\alpha = 0.5$	23.82±0.12	22.04±0.32	21.82±1.63	31.48±0.12	15.87±0.20	**31.39±0.54**

TABLE III: Comparison of training time and communication overhead.

Hetero. Settings	Accuracy (%)	DFedAvgM		D-PSGD		LD-SGD		DFedSAM		DFedPGP		MMDFL	
		Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.
IID	50	672s	20979M	414s	239766M	1651s	26973M	589s	11988M	743s	20957M	**319s**	**746M**
	55	N/A	N/A	1048s	599415M	N/A	N/A	2193s	44956M	1586s	44909M	**899s**	**2097M**
$\alpha = 1.0$	45	961s	29970M	**528s**	374634M	2098s	35964M	1028s	20979M	N/A	N/A	761s	**1721M**
	50	4013s	125877M	**1459s**	749269M	7686s	131871M	2337s	47953M	N/A	N/A	2397s	**5544M**
$\alpha = 0.5$	40	864s	26973M	948s	575438M	1745s	29970M	882s	17982M	N/A	N/A	**772s**	**2415M**
	45	3820s	119883M	**1787s**	959064M	10463s	179824M	2051s	41959M	N/A	N/A	2097s	**7043M**

Test Accuracy Comparison. Table II shows the test accuracy comparison on various datasets using different models within an IID scenario and two non-IID scenarios (α=1.0, 0.5). From the table, we can find that compared with the baselines MMDFL can achieve the best inference performance in all cases. Figure 2 shows the training processes of all methods on the CIFAR-10 dataset using ResNet-8. From the figure, we can observe that MMDFL can achieve the best classification performance within both scenarios and the fastest convergence within the IID scenario.



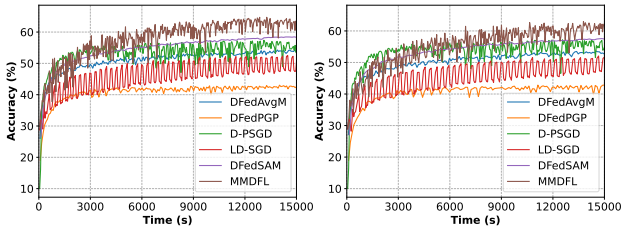
(a) IID (b) $\alpha = 0.5$

Fig. 2: Training processes of MMDFL and five baselines.

Overhead Comparison. For given classification accuracy targets, Table III compares training time and communication overhead between the baselines and our approach on CIFAR-10 using ResNet-8. Here, the second column presents the specified target accuracy information, where the sub-columns "Time" and "Comm." indicate the training time and communication overhead, respectively. We use the notion "N/A" to denote that the target classification accuracy cannot be achieved by using the specified DFL method. From this table, we can find that MMDFL significantly reduces the communication overhead within both IID and non-IID scenarios. Compared to DFedAvgM, MMDFL can reduce communication overhead by about 28 times and accelerate training speed by about 2.1 times in IID scenarios. For example, when dealing with the IID scenario with a target classification accuracy of 50%, MMDFL can reduce 96% communication overhead and reduce 53% training time.

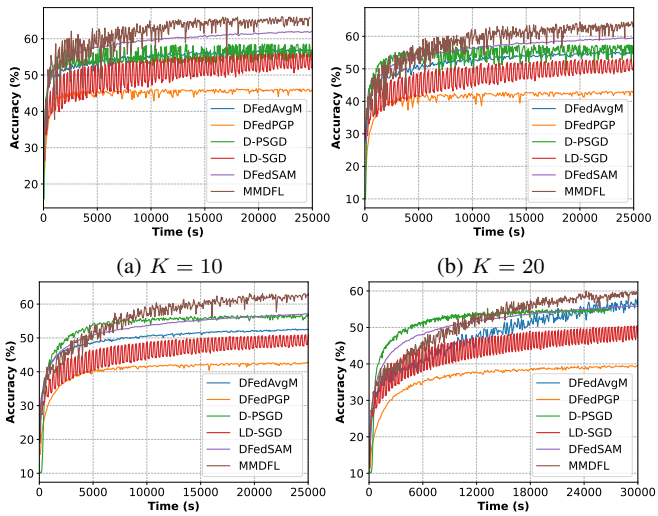
C. Impacts of Different Configurations

Proportions of Devices and Communication Channels. We compared MMDFL with the five baseline methods using the following two configurations: i) **Conf1**, where the weak, middle, strong devices and low, middle, high network connections between devices are both in the proportion 2:3:5; and ii) **Conf2**, where the weak, middle, strong devices and low, middle, high network connections between devices are both in the proportion 5:3:2. Figure 3 shows the training processes of all methods on CIFAR-10 within a non-IID scenario (α=1.0) using ResNet-8. From the figure, we can find that MMDFL always achieves the best test accuracy in both configurations. Meanwhile, MMDFL achieves faster convergence when the proportions of strong devices and high-quality connections are higher.



(a) Conf1 (b) Conf2

Fig. 3: Training processes with different proportions.



(a) $K = 10$ (b) $K = 20$

(c) $K = 40$ (d) $K = 80$

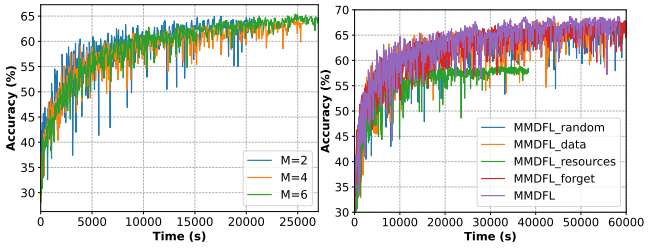
Fig. 4: Training processes with different numbers of devices.

Number of Participating Devices. To evaluate the scalability of MMDFL, we conducted experiments considering different numbers of participating devices, i.e., $K = 10$, 20, 40, and 80, respectively, on CIFAR-10 using ResNet-8. In this experiment, we set the number of traveler models to 20% of the total number of devices. Figure 4 compares MMDFL with the five baselines within a non-IID scenario (α=1.0). We can find that MMDFL achieves the highest classification accuracy regardless of the numbers of devices.

D. Ablation Study

Impact of the Number of Traveler Models. To evaluate the impact of different numbers of traveler models, we considered three different values (i.e., 2, 4, and 6) for the hyperparameter M, respectively. Figure 5(a) shows the experimental results on CIFAR-10 within a non-IID scenario (α=1.0). From the figure, we can find that the impact of using different numbers of traveler models on classification accuracy is negligible. Note that an increase in the number of travelers will result in a longer training time.

Impact of Device Selection Strategies. To evaluate the impact of various device selection strategies, we considered four variants of MMDFL, i.e., "random", "data", "resource", and "forgetting". Specifically, "random" denotes the case where MMDFL selects devices randomly, while "data", "resource", and "forgetting" represent the cases where MMDFL considers only data-wise, resource-wise, and forgetting-wise scores for device selection, respectively. Figure 5(b) shows the ablation study on the CIFAR-10 dataset within a non-IID scenario (α=1.0) using ResNet-8. From the figure, we can find that, compared with variant "random", variants "data" and "forgetting" achieve higher classification accuracy. For the variant "data", since the data-wise score enables traveler models to be trained on a more uniform data distribution, the problem caused by data heterogeneity can be significantly mitigated. For variant "forgetting", since the forgetting-wise score ensures that traveler models can review the data on each device, it enables full utilization of all local data, thus increasing the classification accuracy. Meanwhile, we can find that the variant "resource" spends much less training time than the other variants when their number of communication rounds is the same. However, since it blindly selects devices that complete training quickly, it achieves much lower classification accuracy than the other variants. Note that by combining the advantages of other variants, our proposed MMDFL achieves the highest test accuracy.

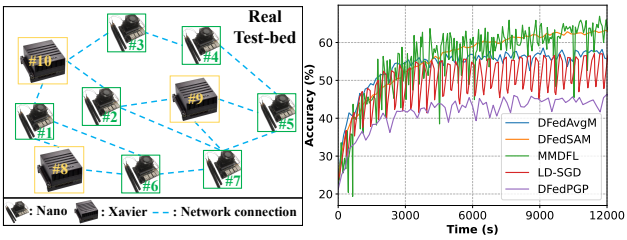


(a) # of traveler models (b) Device selection strategy

Fig. 5: Ablation study of MMDFL.

E. Evaluation on A Real Testbed

We implemented our MMDFL method on top of a real testbed, consisting of three Jetson Xavier compute modules and seven Jetson Nano compute modules. Based on this testbed, we conducted experiments on CIFAR-10 within a non-IID scenario (α=1.0) using RseNet-8, where we set the number of traveler models to 2. Figure 6 presents the network topology of AIoT devices used in our experiment and the comparison results obtained from our real testbed. Since Jetson Nano does not satisfy the memory requirement for D-PSGD training, we did not include the results of D-PSGD here. From this figure, we can observe that MMDFL achieves higher classification accuracy compared to the other four baseline methods.



(a) Network topology (b) Training processes

Fig. 6: DFL training on a real AIoT testbed.

V. Conclusion

Although Decentralized Federated Learning (DFL) has been increasingly used as a promising collaborative learning paradigm in AIoT design, due to the existence of various resource-constrained devices, it suffers from the problems of low classification accuracy and long training time. To address these issues, this paper presents a novel muti-model-based DFL method named MMDFL, which can effectively decouple the relations of neighboring devices for FedAvg operations. Specifically, MMDFL adopts a set of traveler models, which individually traverse the connected neighboring devices for local training and conduct knowledge aggregation on their encounters. Based on our proposed device selection strategy, MMDFL can maximize the learning potential of these travelers and accelerate the overall DFL training process. Comprehensive experiments show the superiority of MMDFL over state-of-the-art DFL methods.

VI. Acknowledgement

This work was supported by Natural Science Foundation of China (62272170) and Shanghai Trusted Industry Internet Software Collaborative Innovation Center. Mingsong Chen is the corresponding author (mschen@sei.ecnu.edu.cn).

REFERENCES

[1] K. Bhardwaj, W. Chen, and R. Marculescu, "New directions in distributed deep learning: Bringing the network at forefront of IoT design," in *Proc. of Design Automation Conference (DAC)*, 2020, pp. 1–6.

[2] J. Zhang and D. Tao, "Empowering things with intelligence: A survey of the progress, challenges, and opportunities in artificial intelligence of things," *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 7789–7817, 2021.

[3] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for internet of things: Recent advances, taxonomy, and open challenges," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1759–1799, 2021.

[4] M. Hu, E. Cao, H. Huang, M. Zhang, X. Chen, and M. Chen, "AIoTML: A unified modeling language for aiot-based cyber-physical systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 11, pp. 3545–3558, 2023.

[5] Z. Gao, A. Li, Y. Gao, B. Li, Y. Wang, and Y. Chen, "FedSwap: A federated learning based 5g decentralized dynamic spectrum access system," in *Proc. of International Conference On Computer-Aided Design (ICCAD)*, 2021, pp. 1–6.

[6] X. Xiao, M. Duan, Y. Song, Z. Tang, and W. Yang, "Fake node-based perception poisoning attacks against federated object detection learning in mobile computing networks," in *Proc. of Design Automation Conference (DAC)*, 2024, pp. 165:1–165:6.

[7] S. Chakraborty, M. A. A. Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Design & Test*, vol. 33, no. 4, pp. 92–108, 2016.

[8] Y. Wu, D. Zeng, Z. Wang, Y. Sheng, L. Yang, A. J. James, Y. Shi, and J. Hu, "Federated contrastive learning for dermatological disease diagnosis via on-device learning," in *Proc. of International Conference On Computer-Aided Design (ICCAD)*, 2021, pp. 1–7.

[9] J. Pan, X. Lin, J. Xu, Y. Chen, and C. Zhuo, "Lithography hotspot detection based on heterogeneous federated learning with local adaptation and feature selection," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 43, no. 5, pp. 1484–1496, 2024.

[10] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: an efficient federated learning framework for heterogeneous mobile clients," in *Proc. of Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2021, pp. 420–437.

[11] X. Zhang, M. Hu, J. Xia, T. Wei, M. Chen, and S. Hu, "Efficient federated learning for cloud-based AIoT applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 11, pp. 2211–2223, 2021.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.

[13] J. Shi, Y. Wu, D. Zeng, J. Tao, J. Hu, and Y. Shi, "Self-supervised on-device federated learning from unlabeled streams," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 12, pp. 4871–4882, 2023.

[14] Z. Chen, C. Jia, M. Hu, X. Xie, A. Li, and M. Chen, "Flexfl: Heterogeneous federated learning via apoz-guided flexible pruning in uncertain scenarios," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 4069–4080, 2024.

[15] J. Xia, T. Liu, Z. Ling, T. Wang, X. Fu, and M. Chen, "PervasiveFL: Pervasive federated learning for heterogeneous iot systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 11, pp. 4100–4111, 2022.

[16] R. Chandrasekaran, K. Ergun, J. Lee, D. Nanjunda, J. Kang, and T. Rosing, "FHDnn: Communication efficient and robust federated learning for AIoT networks," in *Proc. of Design Automation Conference (DAC)*, 2022, pp. 37–42.

[17] Y. Cui, K. Cao, J. Zhou, and T. Wei, "HELCFL: High-efficiency and low-cost federated learning in heterogeneous mobile-edge computing," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1227–1232.

[18] T. Sun, D. Li, and B. Wang, "Decentralized federated averaging," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 45, no. 4, pp. 4289–4301, 2023.

[19] H. Ye, L. Liang, and G. Y. Li, "Decentralized federated learning with unreliable communications," *IEEE Journal of Selected Topics in Signal Processing*, vol. 16, no. 3, pp. 487–500, 2022.

[20] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, S. L. Bernal, G. Bovet, M. G. Pérez, G. M. Pérez, and A. H. Celdrán, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Communications Surveys and Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023.

[21] C. Jia, M. Hu, Z. Chen, Y. Yang, X. Xie, Y. Liu, and M. Chen, "AdaptiveFL: Adaptive heterogeneous federated learning for resource-constrained AIoT systems," in *Proc. of Design Automation Conference (DAC)*, 2024, pp. 84:1–84:6.

[22] L. Wang, S. Xu, X. Wang, and Q. Zhu, "Addressing class imbalance in federated learning," in *Proc. of AAAI Conference on Artificial Intelligence (AAAI)*, 2021, pp. 10 165–10 173.

[23] M. Hu, Z. Xia, D. Yan, Z. Yue, J. Xia, Y. Huang, Y. Liu, and M. Chen, "GitFL: Uncertainty-aware real-time asynchronous federated learning using version control," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2023, pp. 145–157.

[24] Y. Shi, L. Shen, K. Wei, Y. Sun, B. Yuan, X. Wang, and D. Tao, "Improving the model consistency of decentralized federated learning," in *Proc. of International Conference on Machine Learning (ICML)*, 2023, pp. 31 269–31 291.

[25] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proc. of International Conference on Machine Learning (ICML)*, 2019, pp. 344–353.

[26] C.-Y. Huang, K. Srinivas, X. Zhang, and X. Li, "Overcoming data and model heterogeneities in decentralized federated learning via synthetic anchors," in *Proc. of Int. Conf. on Machine Learning (ICML)*, 2024.

[27] Y. Liao, Y. Xu, H. Xu, L. Wang, C. Qian, and C. Qiao, "Decentralized federated learning with adaptive configuration for heterogeneous participants," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, pp. 7453–7469, 2024.

[28] J. Liu, J. Liu, H. Xu, Y. Liao, Z. Wang, and Q. Ma, "YOGA: Adaptive layer-wise model aggregation for decentralized federated learning," *IEEE/ACM Transactions on Networking (TON)*, vol. 32, no. 2, pp. 1768–1780, 2024.

[29] Y. Liu, Y. Shi, B. Wu, Q. Li, X. Wang, and L. Shen, "Decentralized directed collaboration for personalized federated learning," in *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 23 168–23 178.

[30] S. Chen, Y. Xu, H. Xu, Z. Ma, and Z. Wang, "Enhancing decentralized and personalized federated learning with topology construction," *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9692–9707, 2024.

[31] L. Wang, Y. Xu, H. Xu, M. Chen, and L. Huang, "Accelerating decentralized federated learning in heterogeneous edge computing," *IEEE Transactions on Mobile Computing (TMC)*, vol. 22, no. 9, pp. 5001–5016, 2023.

[32] Z. Tang, S. Shi, B. Li, and X. Chu, "GossipFL: A decentralized federated learning framework with sparsified and adaptive communication," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 34, no. 3, pp. 909–922, 2023.

[33] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F. Li, "ImageNet: A large-scale hierarchical image database," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, 2015.

[37] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[38] X. Lian, C. Zhang, H. Zhang, C. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. of Annual Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 5330–5340.

[39] X. Li, W. Yang, S. Wang, and Z. Zhang;, "Communication-efficient local decentralized SGD methods," *arXiv/1910.09126*, 2015.

Automated Generation of Decoders for Irregular Instruction Sets Using Information-Theoretic Decision Tree Construction Algorithms

Lillian Tadros
Robotics Research Institute
Technical University of Dortmund
Email: lillian.tadros@tu-dortmund.de

***Abstract*—Instruction decoders are indispensable components of the System-on-Chip design flow and major constituents of instruction set simulators and processor toolchains. The complex and lengthy process of manual decoder design can be greatly alleviated by automated decoder generation tools based on high level instruction definitions. Unfortunately, automatic generation is challenged by the rising complexity of instruction sets as well as irregularities such as non-uniform opcodes, logic propositions on bit fields and multiple or nested specializations. The few available state-of-the-art decoder generation tools either cannot handle irregularities altogether or produce inadequate results, either functionally or w.r.t. performance. Moreover, they are largely ad hoc and do not bear on any of the well-established work on decision tree generation. This paper presents a sophisticated decision-tree algorithm for the problem of generating decoders for irregular instruction sets. Our algorithm has produced fully automated, functionally correct and cost-aware decoders for the SPARC, MIPS32 and ARMv7 instruction sets. Our results prove the application of information-theoretic concepts to decoder generation a most promising approach.**

I. Introduction

One of the central components in SoC prototyping is the instruction decoder, which is responsible for classifying instructions given the corresponding machine code as defined by an instruction set architecture (ISA). Decoders typically appear as components of different toolchain elements, virtual machines, or instruction-set simulators (ISS). Their range of use covers the whole design cycle, starting with HW/SW co-design, simulation, verification or system support tools. This wide utilization range, coupled with the relative longevity of ISA evolving over time and over microarchitectures sharing the same ISA, makes it highly rewarding to invest in an optimized decoder model that can be repeatedly deployed, modified or expanded.

Manual decoder modeling, however, has reached unprecedented complexity with the recent trend towards more sophisticated ISAs, often encompassing several hundred instructions, numerous instruction formats, multiple sets of extensions (SIMD and others), and sometimes special modes with different instruction lengths (e.g. ARM Thumb and others). To further complicate matter, new instructions are often defined using complex propositions on multiple bit fields, rendering the classical notion of a cleanly defined opcode obsolete, and requiring a completely novel approach towards modeling and manipulating encodings. To date, all current decoder generation algorithms either cannot handle such instruction set irregularities, or claim to do so but fail at corner cases, producing incorrect decoders (see Section II). Surprisingly, they do not at all pertain to the substantial body of work on decision trees, but instead offer rudimentary, use-case-specific solutions. From our point of view, the more complex the ISA, the greater the need for an automated, functionally sound, robust, and optimized decoder generation algorithm.

In this paper, we present an algorithm for generating efficient decoding trees for complex and irregular instruction sets based on information-theoretic concepts. The paper is organized as follows: Section II discusses relevant related work. We proceed with the problem statement and required definitions in Section III. We present and experimentally evaluate our decoding algorithm in Sections IV and V, respectively. We conclude and outline future work in Section VI.

II. Related Work

The initial attempt to achieve fully-automated decision tree generation was made by Theiling [1], which tests all *defined* (i.e. set to either zero or one) bits in the current instruction subset, and assigns the instructions to child nodes according to their mask value. The algorithm does not consider frequencies of instruction occurrence or tree cost, which can, in the extreme case, degenerate into a shallow but correspondingly wide table with many memory-consuming branches. Most importantly, the algorithm can neither handle non-uniform ISAs nor propositions on instruction fields (see Section III-B), and would thus fail on ISAs such as ARM (version $\geq$ v6) or MIPS (version $\geq$ R6). Nevertheless, the algorithm initiated the study of fully automatic decoder generation and was reproduced in other work [2].

The PART algorithm by Fournel at al. [3] expands Theiling's work to handle logic propositions. The authors define propositions in terms of binary decision diagrams (BDD), which are then expanded to all valid encodings. The methodology retains all of the original algorithm's shortcomings, as well as adding a highly inefficient preprocessing step which involves exponential expansion of the instruction set.

Later work by Okuda et al. [4] also builds upon Theiling's work by endeavoring to handle non-uniform ISAs and propositions. In contrast to Fournel, propositions are treated as first-class entities and are included as possible splitting criteria. The flaws with respect to performance are retained from the original algorithm and are even exacerbated w.r.t. functionality: The formalism for instruction definition lacks the expressiveness required by many modern ISAs (including even the aesthetic and spartan MIPS), such as inequalities or relations between instruction fields. Furthermore, the algorithm attempts to handle non-uniform ISAs by resorting to propositions, which is not guaranteed to provide a solution or, even worse, may even create functionally erroneous decoders.

Qin et al. [5] addresses some of the deficiencies in Theiling's algorithm by handling non-uniform ISAs, allowing splitting on undefined bits, assigning a heuristic to memory consumption to limit the number of branches, and addressing probabilities by estimating the height of a best-case Huffman subtree. The search space is reduced by imposing contiguity on the chosen masks. Notwithstanding, the algorithm is markedly more compute-intensive than Theiling. Surprisingly, as the measurements in [6] show, these attempts at optimality fail in improving the performance. On the contrary, Qin's algorithm performs slightly worse. As discussed in this survey work, this could well be due to the restriction on mask contiguity, which probably leads to excluding more efficient decoders, as well as to the adopted cost model, which is shown to be irrelevant in regard to the real performance [7]. Likewise, the heuristic for memory consumption is highly disputable.

The EFF algorithm by Fournel [3] attempts to handle propositions by applying Qin's algorithm to a BDD representation of the instruction. By definition, however, Qin's bitwise construction of the decision function must necessarily fail on all propositions larger

than two bits. The authors vaguely mention "helping" the decoder with "special pattern[s]", i.e. presumably hardcoding all propositions which the algorithm would not find otherwise.

Later work by Wagstaff [8] falls back to non-optimal bitwise mask construction. An algorithm by Solovev [9] does not rely on detailed instruction definitions, but rather attempts to reconstruct the instruction encodings from a combination of generic format descriptions, assembly and machine code, which necessarily requires large training data.

Though [10] offers functionally sound decoders for non-uniform ISAs with complex conditionals, it remains ad-hoc and somewhat haphazard. With the non-optimized solution showing better performance than the optimized, there clearly remains a need for a more rigorous approach.

In summary, the inadequacy of existing automatic decoder generators is perhaps best attested by the following two indicators: First, the ongoing research on refinements of and alternatives to interpreted simulation, including caching, static, hybrid, just-in-time and dynamic compilation [11], which, nonetheless, fail to provide the level of both flexibility and accuracy offered by interpreted simulation. Second, after decades of research invested in automatic decoder generation, manual decoders remain as extant, relevant, and ubiquitous as ever [12]–[14].

III. Background and Definitions

A. Instructions, Encodings and Decision Trees

An *instruction instance* or bit string s is a string over the binary alphabet. An *instruction encoding* p is a string over the ternary-logic alphabet, which includes *defined* (zero or one) as well as *undefined* or "don't-care" bits. Bit extraction, slicing, and concatenation in most-significant-bit (MSB) order are denoted by $p[i]$, $p[i{:}j]$, and $p[i]{:}p[j]$, respectively. An ISA defines a set E of *encoding entries* e, which assign a *label* l to each encoding. With the aid of benchmarking, an encoding entry can be extended with a *probability of occurrence* o based on how often instructions pertaining to this encoding are encountered.

The purpose of a *decision tree* is to relate a given instruction instance to the correct encoding. This process is called *matching* and denoted $s \in p$. Internal nodes contain *decision functions* f, which direct the instruction in question to the correct branch. The decision tree leaves are tagged with instruction labels. These definitions assume a *well-formed* ISA, i.e. where each string can only match one encoding.

B. Properties of Instruction Sets

TABLE I: Example irregular instruction set.

	Encoding	Feature
A	`[= 11-- 1---]`	Non-uniform opcodes (A, B, C, D)
B	`[= 0-1- ---0]`	Non-identification bit $B[0]$
C	`[= -00- 0---]`	
D	`[= 10-- 1---]` ∧`[~ ---- -111]`	Boolean proposition
AA	`[= 11-- 111-]`	Specialization of A
AB	`[= 11-- 10-1]`	Another specialization of A
AAA	`[= 11-- 1111]`	Nested specialization of A via AA
E	`[= 0101]`	Non-uniform instruction length

Table I. The instruction encodings are defined using a notation where the bits are listed in MSB order, defined bits are written as '1' or '0', and undefined bits as '-'. Instruction D contains an exclusion condition: The three least-significant bits cannot equal 111_2. See note on notation in Section IV-A and Table II.

Table I illustrates some of the challenging features of modern instruction sets, such as *non-identification bits*, *specializations* and *non-uniform instruction sizes*. While these three are relatively simple to address, the following two properties are more involved and remain poorly handled by current generator tools:

a) Non-uniform opcodes: An opcode is the smallest set of bits that are sufficient to uniquely identify an instruction. Traditionally, opcodes consisted of the same adjacent set of bits in all instructions. Thus, simply masking the opcode field was enough for classifying instructions. In modern instruction sets this is hardly the case. In order to expand the ISA while retaining some backward compatibility, existing encoding formats are often expanded with sub-opcodes, special opcodes, and so on. This has two effects on decoding: First, the number of bits required for identifying instructions generally grows. In the worst case, it can degenerate to all n instruction bits, instead of the lower limit of $\lceil log_2(m) \rceil$ that are strictly required to classify m encodings. In Table I, decoding instructions A, B, C and D requires checking bits 3, 5, 6 and 7. Second, since different formats define opcodes anywhere over the whole instruction, leaving others undefined, it can often occur that there is no single bit that is defined for all encoding formats. This is the case for instructions A, B, C and D in Table I, which do not share a single common defined bit. If an algorithm can only test defined bits, it will fail on valid ISAs, as does Theiling's algorithm discussed above. Other algorithms solve such cases of non-uniform opcodes by *duplication*: If some encodings do not yield a unique result of the chosen decision function, they are assigned to all matching branches.

b) Propositions: Traditional encoding definitions are simple conjunctions of bits. More recent ISAs make use of propositions, which entail more complex restrictions on bit fields. The simplest propositions are negated conjunctions, such as in instruction D in Table I. In general, propositions lead to rejecting some strings that would otherwise match an encoding. As with other parts of an encoding, propositions can either be required for identification or at least for correctness. The proposition on instruction D pertains to the latter case.

Although we chose to analyze three RISC ISAs, none of them exhibits trivial encoding complexity. The simplest was SPARC [15], which avoided propositions and non-uniform opcodes, but still made use of other complex features. The other two, ARMv7 [16] (including floating-point and SIMD instructions), and MIPS R6 [17] boast almost all of the above-mentioned properties.

IV. Decoder Construction

A. Formalizing Encodings

The definitions provided so far are bit-oriented and not tailored to handle propositions. The propositions in both ARMv7 and MIPS32 can involve either equalities, *dis*equalities or inequalities, all of which can be defined either between a field and a constant or between two fields. We propose a formalism which we illustrate in Table II. Each square bracket is a predicate that applies the operator inside the brackets to the indicated bits. An operation between two fields is represented using letters to mark each field. The first line typically corresponds to a "classical" encoding definition, i.e. a conjunction of bits.

It is easy to see that our representation is a shorthand for first-order (predicate) logic statements in conjunctive normal form, without quantification and with predicates in the binary domain. As such, each encoding is written as a conjunction of disjunctions of predicates. The MIPS BEQZALC encoding pattern can thus be rewritten as $\neg p[31] \wedge \neg p[30] \wedge p[29] \wedge \neg p[28] \wedge \neg p[27] \wedge \neg p[26] \wedge (p[20{:}16] \neq 00000_2) \wedge (p[25{:}21] \geq p[20{:}16])$. We formalize our representation by the following BNF:

```
encoding          ::= clause ( ∧ clause )*
clause            ::= predicate
                    | ( ( predicate ( ∨ predicate )+ ) )
predicate         ::= [ op literal-or-field+ ]
op                ::= = | ~ | < | > | <= | >=
literal-or-field  ::= 0 | 1 | - | A | B | ...
```

TABLE II: Examples encodings in ARMv7 and MIPS32 R6.

LABEL	ENCODING

ARMv7 ADD (imm)	`[= ---- 0010 100- ---- ---- ---- ---- ----]`
	`∧ [~ 1111 ---- ---- ---- ---- ---- ---- ----]`
	`∧ [~ ---- ---- ---- 1101 ---- ---- ---- ----]`
	`∧ [~ ---- ---- ---0 1111 ---- ---- ---- ----]`
	`∧ [~ ---- ---- ---1 ---- 1111 ---- ---- ----]`

ARMv7 RSB (reg)	`[= ---- 0000 011- ---- ---- ---- ---0 ----]`
	`∧ [~ 1111 ---- ---- ---- ---- ---- ---- ----]`
	`∧ ( [= ---- ---- ---0 ---- ---- ---- ---- ----]`
	`∨[~ ---- ---- ---- ---- 1111 ---- ---- ----])`

MIPS32 BEQZALC	`[= 001000 ----- ----- ---- ---- ---- ----]`
	`∧ [~ ------ ----- 00000 ---- ---- ---- ----]`
	`∧ [>=------ SSSSS TTTTT ---- ---- ---- ----]`

Table II. The table lists some encodings from ARMv7 and MIPS32 R6 using the proposed notation in MSB order. Each predicate applies the operator inside the bracket to the encoding and the bracket value. The second line can thus be interpreted as $p[31{:}28] \neq 1111_2$. In the case of predicates between two fields, each field is designated using a letter, and assigned to the operator in reading order. The last line is e.g. $p[25{:}21] \geq p[20{:}16]$. The first line in each encoding is usually a simple conjunction of bits and corresponds to "classical", proposition-free encodings.

Our formalism suffices to describe the full ISA of our three platforms. The primary advantage is that there is generally no exponential expansion of predicates involved, which would be the case if we attempted a conversion to zeroth-order (propositional) logic. Furthermore, not requiring bit field contiguity generally simplifies the notation. Most importantly, the notation abstracts predicates from their bit constituents and makes them first-class citizens that are on equal footing with single bits. As will be apparent presently, this opens up new possibilities in decision tree construction.

B. Decision Functions

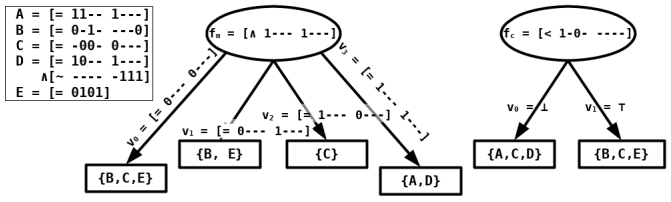


Fig. 1: Two example decision functions applied to the instruction set in Table I, where subinstructions are automatically assigned to all branches containing the parent and therefore not explicitly handled. N-ary functions such as f_m can only apply one type of operator: the ternary-logic conjunction operator $\wedge$, and result in at most 2^J branches for $|J{=}defined(f_m)|$. The values are attached to the branches and specify the results of applying the decision function. Binary functions such as f_c can apply any relational operator (in this case: less-than), and result in two branches with boolean truth values. Our model of computation interprets defined bits in MSB order and ignores undefined ones, so that f_c is interpreted as $p[7]{:}p[5] \leq 10_2$.

We define two types of decision functions:

a) Mask functions: f_m are n-ary functions that first select a set of instruction bits via a given mask, then assign the instruction to the matching branch depending on the result. Formally, they apply some string over the alphabet $\{1, -\}^n$ using the conjunction operator $\wedge$. A mask where $|defined(f_m)|{=}J$ results in 2^J *values* $v_j{\in}\{0, 1, -\}^n$, which are used to tag the corresponding child branches. The values v_j are written using a restricted form of the encoding formalism of the previous section, which is a single equality predicate of the form `[= ...]`.

b) Condition functions: f_c are binary functions that compare the instruction instance to a fixed value using relational operators such as equality, *dis*equality or inequality, resulting in two (matching and *un*matching) branches, tagged with $v_1{=}\top$ and $v_0{=}\bot$, respectively. Compared to masks, binary functions are useful when it is more efficient to split exactly one instruction away from the set. They can be regarded as a specialization of masks, where one branch value is kept (matching) and all others subsumed in the *un*matching branch. A second advantage is that, unlike bit-oriented masks, they can handle conditions. As mentioned in the previous section, we treat both bits and conditions as atoms. In fact, each predicate contained within a clause, within a term, is treated as an atom, and is thus considered a possible splitting function. To be as general as possible, we allow condition functions to take any form that complies to the complete grammar of Section IV-A. This enables us to split off any instruction from the current set by simply setting its complete encoding as the decision function.

Fig. 1 illustrates both function types. Either type utilizes three-bit logic, meaning that the functions can operate on a not necessarily contiguous subset of the instruction bits. The definition of matching will be expanded shortly, but, in general, we say that an instruction matches *a binary function*, but matches *an n-ary function value* as assigned to a given branch.

C. Matching

Matching is relevant to three different contexts: During decoding, an instruction string is successively matched against each decision function ($s{\in}f$), until it is finally matched to an encoding ($s{\in}p$). During *decoder generation*, we start with a set of encodings at the root, which is iteratively split by matching an encoding against a decision function ($p{\in}f$). This generic definition of matching is made possible by our formalism, which models instructions, encodings and decision functions alike. Without loss of generality, we concentrate on the third context in the following discussion, as it is the most complex case and the one most relevant to decoder generation. An initial, bit-based definition of matching might look thus:

Full match $p \in f$:	$\forall i \in [0, n-1]$	$p[i] = f[i]$	$\vee$	$f[i] = -$	
Partial match $p \,\tilde{\in}\, f$:	$(\forall i \in [0, n-1]$	$p[i] = f[i]$	$\vee$	$f[i] = -$	$\vee \quad p[i] = -)$
	$\wedge \quad (\exists i$	$p[i] = -$	$\wedge$	$f[i] \neq -)$	
*Un*match $p \notin f$:	$\neg(p \in f)$	$\wedge \quad \neg(p \,\tilde{\in}\, f)$			

These definitions are for binary functions. For mask functions, $p{\in}v_j$, $p\tilde{\in}v_j$ and $p{\notin}v_j$ are defined accordingly. These definitions cannot handle conditions. The examples below illustrate how the interplay of bits and conditions can affect the matching result for both decision function types. As shown, conditions can change a partial or a full match one into the other, or either to an *un*match:

f_m = `[= 11-- 11--]`, v_1 = `[= 00-- 01--]`
p_1 = `[= 00-- ----] ∧ [> ---- 01--]` $\models p_1 \notin v_1$
p_2 = `[= 00-- 0---] ∧ [> ---- -010] ∧ [~ ---- --11]` $\models p_2 \in v_1$

f_{c1} = `[< ---- 10--]`
p_3 = `[= 00-- 1---]` $\models p_3 \notin f_{c1}$
p_4 = `[= 00-- ----] ∧ [> ---- 01--]` $\models p_4 \notin f_{c1}$
p_5 = `[= 00-- ----] ∧ [~ ---- 1---]` $\models p_5 \in f_{c1}$
p_6 = `[= 00-- 0---] ∧ [~ ---- -1--]` $\models p_6 \in f_{c1}$

f_{c2} = `[= 00-- ----] ∧ [~ ---- 00--]`
p_7 = `[= 00-- ----] ∧ [< ---- 11--]` $\models p_7 \,\tilde{\in}\, f_{c2}$
p_8 = `[= 00-- ----] ∧ [< ---- 01--]` $\models p_8 \notin f_{c2}$

We address such cases by a new definition of matching that is in line with our formalism of IV-A. Informally, a full match is when all clauses set by the decision function are fulfilled, i.e. implied by, the encoding. An *un*match is when the encoding contradicts the decision function. And a partial match is when neither case is true. This can be formalized using either logic- or set-theoretic concepts:

Full match $p \in f$:	$p \rightarrow f$	f	$\subseteq \; p$
Partial match $p \,\tilde{\in}\, f$:	$\exists x(x{\in}p \wedge x{\in}f) \wedge \exists y(y{\in}p \wedge y{\notin}f)$	f	$\not\subseteq \; f \wedge p \cup f \neq \phi$
*Un*match $p \notin f$:	$p \rightarrow \neg f$	f	$\subseteq \; p'$

D. Base Algorithm

Building the decoding tree is a recursive process described in pseudocode in Algorithm 1.

Lines 2-4 are the terminal condition of the recursion, where the set of encoding entries contains only one entry p_0. The node is thus labeled with the encoding name or identifier.

Algorithm 1 Base Algorithm

```
    ▷ Remove subinstr, from encoding entry set E and attach to parents.
 1: function MAKETREE(node, E)
        ▷ Terminal Condition:
 2:     if |E| = 1 then
 3:         if e0.encoding ⊆ node.cum then
 4:             node.label ← e0.label
            ▷ Create final node with all remaining untested bits and conditions:
 5:         else
 6:             node.func ← e0.encoding − node.cum
 7:             child ← CreateNode({e0})
 8:             child.cum ← CumFunc(node.cum, node.func, v=1)
 9:             node.AddChild(child, v=1)
10:             MakeTree(child, {e0})
11:         end if
12:         return
13:     end if
        ▷ Recursive Case:
14:     (f, {(child_j, v_j, E_j) | 0≤j<J}) ← SelectFunction(node, E)
15:     node.func ← f
16:     for (child_j, v_j, E_j) ∈ {(child_j, v_j, E_j) | 0≤j<J} do
17:         child_j.cum ← CumFunc(node.cum, f, v_j)
18:         for e ∈ {E_j} do
19:             e ← Simplify(f, v_j, e)
20:         end for
21:         node.AddChild(child_j, v_j)
22:         MakeTree(child_j, E_j)
23:     end for
24: end function
```

Lines 6-10 specify the last-but-one iteration: To test non-identification clauses (Section III-B), we create an additional node containing any remaining untested bits or conditions.

The functions *CreateNode* and *AddChild* are related to the tree structure and thus not relevant to the discussion. The functions *CumFunc* and *Simplify* that build the cumulative decision function and simplify the encodings, respectively, will be described in Section IV-E. The lines 14-23 describe the recursive case. The *SelectFunction* is the heart of the algorithm, the details of which we will discuss in Section IV-F. The procedure examines several decision functions, performs the matching according to Section IV-C, evaluates the functions, then returns the selected function f along with a set of children. Each child node $child_j$ is linked to a value v_j that labels the corresponding branch, as well as to its set of matching encodings E_j.

E. Post-Matching

We need to keep track of the sequence of tested decision functions for two reasons: First, to ensure that each encoding is fully tested. Second, during decoder generation, the set of eligible decision functions can be reduced by subtracting all the previously tested. For masks, we simply store the conjunction of all tested bits. Condition functions bits, on the other hand, cannot, in general, be considered tested, except for the matching branch. The *un*matching branch of e.g. f_c = `[= 1111 ----]` only excludes one possibility out of 16, and thus requires further testing of the same bits. The only case when the *un*matching branch is considered tested is when the function involves only one bit. In general, if the function application yields true for exactly one combination of bit values, then these bits are considered tested and added to the cumulative decision function.

After matching, all child encodings are examined and simplified against the decision function. The following MIPS32 encodings illustrate why this is necessary:

```
BOVC          =  [= 001000 ----- ----- ---- ---- ---- ----]
              ∧  [>=------ xxxxx yyyyy ---- ---- ---- ----]
BEQZALC       =  [= 001000 00000 ----- ---- ---- ---- ----]
              ∧  [~ ------ ----- 00000 ---- ---- ---- ----]
f_c           =  [= ------ 00000 ----- ---- ---- ---- ----]
v1 = ⊤,       BOVC, BEQZALC ∈ v1:
BOVC          =  [= 001000 00000 00000 ---- ---- ---- ----]
BEQZALC       =  [= 001000 00000 ----- ---- ---- ---- ----]
              ∧  [~ ------ ----- 00000 ---- ---- ---- ----]
```

It is easy to see that, if BOVC is *not* simplified after applying f_c, the two encodings would be indistinguishable. Generally, simplification involves two aspects: Writing out bits that are set by the decision function but undefined by the encoding, and/or simplifying encoding conditions. The latter step could, in turn, simplify the condition up to the point where we can write out the condition as set bits. Furthermore, if any newly written bits intersect with any conditions, the process needs to be repeated. Fig. 2 illustrates several cases.

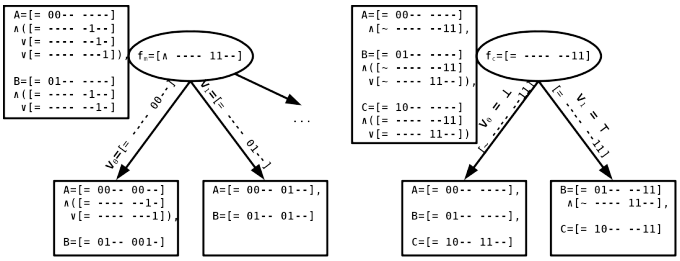


Fig. 2: Simplifying Encodings: Two decision functions applied to the example instruction sets in the top boxes. Both functions involve writing bits as well as simplifying conditions. In general, a false predicate is removed from the enclosing clause (e.g. $(f_c, v_1)(B)$), while a true predicate causes the enclosing clause to be removed (e.g. $(f_c, v_0)(B)$). Simplified clauses can be written out as bits if possible (e.g. $(f_c, v_0)(C)$).

F. Evaluating Decision Functions

Since our aim is to construct efficient decision trees, we need a metric to quantitatively assess decision functions. We agree with the established notion that takes average execution speed as the major cost indicator [5]. An exhaustive solution that generates all possible trees and benchmarks their execution is obviously not feasible. Decision tree literature (e.g. [18]) present many different criteria for top-down evaluation and selection of decision functions at the local level. However, not all of these are applicable or relevant to our use case, so we concentrate on four criteria that especially target classification efficiency and assess their applicability to our use case.

Table III illustrates this calculation stepwise for the example instruction set shown in the upper part of the table. The next part calculates contingency values, which denote the occurrence of a given dependent variable (in decision tree jargon: *class* c) in the presence of a given independent variable (*predictor* or *attribute* a). In our case, the classes are the encoding labels, and the predictors are either single bits or single predicates set to a given attribute-specific value (*category* or *value* $v \in V_a$). Either predictor type results in a binary split according to the matching principles of Section IV-C. A bit predictor that is defined in the encoding leads to the values 1 (full match) and 0 (*un*match), depending on the bit value in the encoding. For example, the encoding A = `1-11` has the value 0 for predictor $p[3] = 0$ (f_m = `[= 1---]`, v_0 = `[= 0---]`), and the value 1 for predictor $p[3] = 1$ (f_m = `[= 1---]`, v_1 = `[= 1---]`). This can be interpreted thus: Should the encoding set be split by the predictor $p[3]$, encoding A would only be assigned to the matching branch. A partial match due to an undefined bit leads to contingency of 0.5, as the encoding is equally likely to occur in either branch. Care must be taken in the presence of conditions. Consider encoding D = `[= --0-] ∧ ([= 1---] ∨ [= ---0])`. Writing out the conditions yields D = `[= 0-00] ∨ [= 1-00] ∨ [= 1-01]`. The predictor $p[3]$, e.g., would then result in D partially matching v_0 and v_1 with a one-third and two-thirds probability, respectively.

The contingency values are then used to calculate the following four metrics. The third part of Table III shows the calculation results for an excerpt of the example instruction set.

a) Chi-Squared: Chi-squared is a classical metric used e.g. in [19] and others. We first calculate the expected value E followed by χ^2 for a given value v and class c combination, then build the sum over all classes and all attribute values. Predictors with a higher χ^2 value are considered better splitting candidates:

TABLE III: Contingency table and decision function metrics for an example instruction set.

Class	Encoding	Patterns	$p(c)$
A	`[= 1-11]`	`[= 1-11]`	0.41
B	`[= 0---]∧[~ --00]∧[~ --11]`	`[= 0-01]∨[= 0-10]`	0.28
C	`[= --1-]∧[~ 0--0]∧[~ 1--1]`	`[= 0-11]∨[= 1-10]`	0.19
D	`[= --0-]∧([= 1---]∨[= ---0])`	`[= 0-00]∨[= 1-00]∨[= 1-01]`	0.12

Bit Contingency Table ($p_v(c)$)

Class	b[3]=0	b[3]=1	b[2]=0	b[2]=1	b[1]=0	b[1]=1	b[0]=0	b[0]=1	$p(c)$
A	0·0.41	1·0.41	0.5·0.41	0.5·0.41	0·0.41	1·0.41	0·0.41	1·0.41	0.41
B	1·0.28	0·0.28	0.5·0.28	0.5·0.28	0.5·0.28	0.5·0.28	0.5·0.28	0.5·0.28	0.28
C	0.5·0.19	0.5·0.19	0.5·0.19	0.5·0.19	0·0.19	1·0.19	0.5·0.19	0.5·0.19	0.19
D	0.33·0.12	0.57·0.12	0.5·0.12	0.5·0.12	1·0.12	0·0.12	0.57·0.12	0.33·0.12	0.12
$p(v)$	0.415	0.585	0.5	0.5	0.26	0.74	0.315	0.685	1

Attribute Metrics

Chi-squared: $\chi'^2(a)=\sum_{v\in V_a} p(v)\cdot\sum_{c\in C}(\frac{(p_v(c)-p(c))^2}{p(c)})$

$\chi'^2(b[1]{:}0) = \frac{(\frac{0\cdot0.41}{0.26}-0.41)^2}{0.41}+\frac{(\frac{0.5\cdot0.28}{0.26}-0.28)^2}{0.28}+\frac{(\frac{0\cdot0.19}{0.26}-0.19)^2}{0.19}+\frac{(\frac{1\cdot0.12}{0.26}-0.12)^2}{0.12}\approx 1.81$

$\chi'^2(b[1]{:}1) = \frac{(\frac{1\cdot0.41}{0.74}-0.41)^2}{0.41}+\frac{(\frac{0.5\cdot0.28}{0.74}-0.28)^2}{0.28}+\frac{(\frac{1\cdot0.19}{0.74}-0.19)^2}{0.19}+\frac{(\frac{0\cdot0.12}{0.74}-0.12)^2}{0.12}\approx 0.224$

$\chi'^2(b[1]) \approx 0.26\cdot 1.81 + 0.74\cdot 0.224 \approx 0.636$

$\chi'^2(a)$	0.695	0	0.636	0.332	

Gini Index: $Gini(a)=\sum_{v\in V_a} p(v)\cdot(1-\sum_{c\in C} p_v(c)^2)$

$Gini(b[1]{:}0) = 1-(\frac{0\cdot0.41}{0.26})^2-(\frac{0.5\cdot0.28}{0.26})^2-(\frac{0\cdot0.19}{0.26})^2-(\frac{1\cdot0.12}{0.26})^2\approx 0.497$

$Gini(b[1]{:}1) = 1-(\frac{1\cdot0.41}{0.74})^2-(\frac{0.5\cdot0.28}{0.74})^2-(\frac{1\cdot0.19}{0.74})^2-(\frac{0\cdot0.12}{0.74})^2\approx 0.591$

$Gini(b[1]) \approx 0.26\cdot 0.497 + 0.74\cdot 0.519 \approx 0.567$

$Gini(a)$	0.472	0.703	0.567	0.599	

Entropy: $IG'(a)=-\sum_{v\in V_a} p(v)\cdot\sum_{c\in C}(p_v(c)\cdot log_2(p_v(c)))$

$IG'(b[1]{:}0) = -\frac{0.5\cdot0.28}{0.26}\cdot log_2(\frac{0.5\cdot0.28}{0.26})-\frac{1\cdot0.12}{0.26}\cdot log_2(\frac{1\cdot0.12}{0.26})\approx 0.998$

$IG'(b[1]{:}1) = -\frac{1\cdot0.41}{0.74}\cdot log_2(\frac{1\cdot0.41}{0.74})-\frac{0.5\cdot0.28}{0.74}\cdot log_2(\frac{0.5\cdot0.28}{0.74})-\frac{1\cdot0.19}{0.74}\cdot log_2(\frac{1\cdot0.19}{0.74})\approx 1.43$

$IG'(b[1]) \approx 0.26\cdot 0.998 + 0.74\cdot 1.43 \approx 1.32$

$IG'(a)$	1.19	1.86	1.32	1.55	

Info Gain Ratio: $H(X)=-\sum_{x\in X}(p(x)\cdot log_2(p(x))$

$SplitInfo(a) = -\sum_{v\in V_a}(p(v)\cdot log_2(p(v))),\qquad IGR(X,a)=\frac{H(X)-IG'(a)}{SplitInfo(a)}$

$H(X) = -0.41\cdot log_2(0.41)-0.28\cdot log_2(0.28)-0.19\cdot log_2(0.19)-0.12\cdot log_2(0.12)\approx 1.86$

$SplitInfo(b[1]) = -0.26\cdot log_2(p(0.26))-0.74\cdot log_2(p(0.74))\approx 0.827$

$IGR(b[1]) \approx \frac{1.86-1.32}{0.827}\approx 0.661$

$IGR(a)$	0.693	0	0.661	0.355	

Table III. The table is divided into three parts. The first part shows four encodings that entail both defined bits and conditions. The second part contains the contingency table using bits as predictors. The class probability is multiplied with values between 0 and 1, where 0 indicates an unmatch, 1 a full match, and all other values a partial match of the corresponding probability. Total values are calculated per bit (predictor, independent variable, column) and encoding label (class, dependent variable, row). In the third part, the four metrics chi-squared, gini index, entropy and information gain ratio are calculated. An example is given per metric to illustrate the calculation. The shaded row shows the metric calculation results for all four bits (predictors). The contingency table for predicates is calculated similarly.

$$E(v,c) = p(c)\cdot incidence(v) = p(c)\cdot\sum_{c\in C} incidence(v,c)$$

$$\chi^2(v,c) = \frac{(incidence(v,c)-E(v,c))^2}{E(v,c)}$$

$$\chi^2(a) = \sum_{v\in V_a}\sum_{c\in C}\chi^2(v,c)$$

We can formulate χ^2 in terms of the probability instead of the incidence, where $p(v)$ is the child node probability relative to the parent, $p(c)$ is the probability of the class within the parent node, and $p_v(c)$ the probability of the class within the child node:

$$\chi^2(v,c) = (\sum_{c\in C} incidence(v,c))\cdot\frac{(p_v(c)-p(c))^2}{p(c)}$$

In the context of decoders, we want a metric that is independent of the number of samples. To achieve this, we can calculate an alternate χ'^2 where we replace the first factor with the probability $p(v)$. Since we are only interested in comparing chi-squared for different attributes, the change in absolute value is of no relevance since the chi-squared ratios of the attributes remain the same:

$$\chi'^2(a) = \sum_{v\in V_a} p(v)\cdot\sum_{c\in C}(\frac{(p_v(c)-p(c))^2}{p(c)})$$

b) Gini Index: The Gini index is another classical metric, used e.g. in [20]. The index is calculated per predictor as follows:

$$Gini(v) = 1-\sum_{c\in C} p_v(c)^2$$

$$Gini(a) = \sum_{v\in V_a}(p(v)\cdot Gini(v)) = \sum_{v\in V_a} p(v)\cdot(1-\sum_{c\in C} p_v(c)^2)$$

The contingency values p are calculated per predictor and class as discussed and the square is summed over all encodings in the encoding set E. Predictors with lower Gini index values are considered better candidates.

c) Entropy: The entropy, as used e.g. in [21] or the information gain of a given encoding subset X is calculated as follows:

$$H(X) = -\sum_{x\in X}(p(x)\cdot log_2(p(x)))$$

The algorithm seeks to maximize the splitting information gain IG, which is the original parent entropy $H(X)$ minus the sum of the resulting entropies of the subsets $H(V_a)$ after splitting on predictor a. The child entropies are modulated with the probability of the child branch relative to all encodings in the current subset:

$$IG(X,a) = H(X) - \sum_{v\in V_a}(p(v)\cdot H(V_a))$$

Equivalently, it suffices to minimize the sum of the modulated child nodes' entropies:

$$IG'(a) = \sum_{v\in V_a}(p(v)\cdot H(V_a)) = -\sum_{v\in V_a} p(v)\cdot\sum_{c\in C}(p_v(c)\cdot log_2(p_v(c)))$$

d) Information Gain Ratio: The information gain ratio, as used e.g. in [22], tries to balance the information gain's bias towards higher child branch numbers by using gain ratio normalization:

$$SplitInfo(a) = -\sum_{v\in V_a}(p(v)\cdot log_2(p(v)))$$

$$IGR(X,a) = \frac{IG(X,a)}{SplitInfo(a)} = \frac{1}{SplitInfo(a)}\cdot(H(X)-\sum_{v\in V_a}(p(v)\cdot H(V_a)))$$

G. Selecting and Combining Complex Decision Functions

As mentioned, the function $SelectFunction(node, E)$ is where the four criteria of Section IV-F are implemented. All four are applied as follows:

- The set of all predictors is constructed. This includes all bits and all predicates defined in any encoding in the current subset. This set is filtered to exclude all bits and predicates that have already been tested. Note that predictors are not whole clauses, but single predicates therein. This offers more flexibility and can be the only solution in some situations.
- The appropriate metric is calculated for each set element.
- Decision function selection offers two variants: In *single mode*, the predictor, either bit or predicate, with the best metric is

		X^2 Multi	Gini Idx Multi	Entropy Multi	I-Gain Ratio Multi	X^2 Single	Gini Idx Single	Entropy Single	I-Gain Ratio Single
SPARC v8	Instructions	214							
	Total Nodes	503	494	1218	2722	560	575	840	1112
	Average Static Depth	2.051	1.965	2.499	2.37	10.028	9.864	9.581	9.827
	Generation Time [s]	0.96	0.997	1.291	1.713	0.35	0.344	0.406	0.522
	Gen. Peak Mem. [MB]	56	55	58	66	56	56	57	58
MIPS32 v6.05	Instructions	213							
	Total Nodes	980	980	3048	14426	583	591	897	1642
	Average Static Depth	2.545	2.545	2.493	2.392	10.962	10.568	10.073	10.385
	Generation Time [s]	31.653	30.322	53	219	48.857	52.759	60.756	102.068
	Gen. Peak Mem. [MB]	184	184	184	207	186	184	184	184
ARMv7-A	Instructions	326							
	Total Nodes	8008	8008	8008	32930	837	828	1539	3059
	Generation Time [s]	86.098	85.727	86.614	103.252	172.705	174.085	125.151	121.135
	Gen. Peak Mem. [MB]	375	393	376	435	371	376	375	374

Fig. 3: Decoder generation statistics.

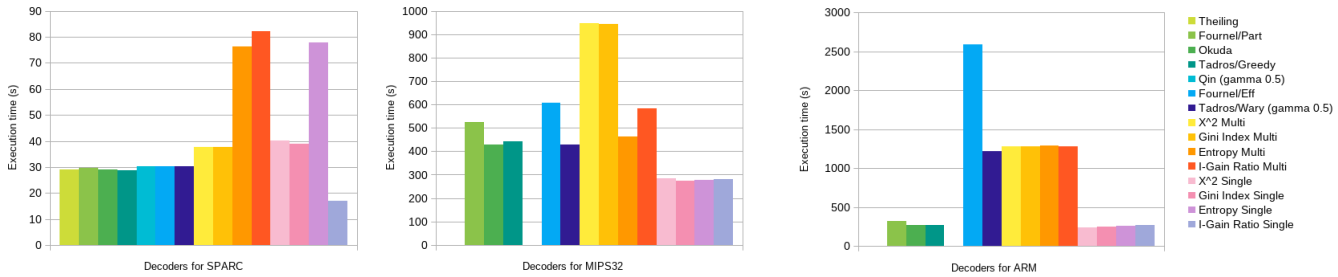


Fig. 4: Decoder execution statistics.

chosen, where 'best' implies maximum or minimum depending on the algorithm. *Multiple mode* also chooses the best single predicate, but handles bits differently: These are first sorted according to their merit. The next best bit is then added to the candidate mask and the encodings split accordingly. As the mask now contains two bits, the result is at most a four-branch tree, for which the metric is again calculated. This process continues iteratively with the next best bit as long as the metric improves and the mask length is within 8 bits to prevent overly broad trees. The resulting J-bit mask is then compared to the best single predicate and the better of the two chosen.

- We refrain from any pruning or postprocessing for better contrast between the different metrics.

H. Decoder Generation

V. Experimental Results

In order to implement our algorithm, we largely rewrote the open-source processor generation tool TRAP-Gen (TRansactional Automatic Processor GENerator) [23] for our purposes. The tool could initially generate a SPARC SystemC processor model from a high-level ISA description in Python 2.7 using Qin's algorithm [5]. Neither MIPS32 nor ARMv7 could be generated using this setup. We thus implemented our algorithm using all four metrics in both single and multiple mode and used it to generate eight decoders for each of the three platforms. Generation statistics are shown in Fig. 3. The three platforms are ordered according to rising complexity, as can be seen from the number of instructions, total number of tree nodes, generation time and peak memory during generation. Single mode generally creates deeper trees (larger "Average static depth") than multiple mode, which, in turn, makes it up in tree breadth (similar "Total Nodes"). As will be obvious in the execution statistics, tree depth is only one of several factors that determine the execution speed.

A. Decoder Execution

As we posit that the runtime decoding speed is the most significant decoder evaluation criteria, we proceeded to measure the execution performance of each decoder. To this end, we first tested all generated decoders for functional correctness, i.e. we ascertained that all instructions are decoded properly. Then, we used the GNU toolchains with no optimization to cross-compile the *mini* version of PolyBench [24], a suite of 30 numerical benchmarks in various application domains such as linear algebra, image processing, physics, dynamic programming, statistics, and others. All 30 benchmarks were then run on QEMU [13] in user mode in combination with the toolchain's GDB, which dumped the instruction machine code to a text file. Next, a decoder test mode was coded in TRAP-Gen, where, instead of executing binaries, a text file is read and the hex code loaded into the memory model, which, in the functional abstraction level used in test mode, is a simple array. The instruction behavior is ignored in test mode, meaning that the instruction is only read from memory, decoded, then discarded. This avoids distorting the results by the overhead of executing the instruction behavior. Code for monitoring the time was also inserted in the decoder C++ implementation just before and immediately after decoding each instruction and summed up for all instructions of a benchmark.

To compare against existing solutions, we also implemented seven other algorithms as published by Theiling [1], Qin [5], Fournel ("Part" and "Eff") [3], Okuda [4] and Tadros ("Greedy" and "Wary") [10]. As neither Theiling nor Qin can handle propositions, they were only applicable to SPARC. Although Fournel and Okuda both attempt to handle propositions, they were either very resource-intensive (Fournel/Part), required hardcoding some instructions (Fournel/Eff), or produced functionally erroneous decoders (Okuda). To be able to roughly compare the performance nevertheless, we followed the approach delineated in [6] to adapt the algorithms so as to produce functionally correct decoders for our chosen platforms. The results are shown in Fig. 4. We grouped and color-coded the algorithms as follows: The first group (green) is composed of greedy algorithms that do not attempt to optimize the tree cost, the second (blue) performs some kind of optimization, the third (yellow to red) is our multiple mode and the fourth (pastel) our single mode. Interestingly, the optimization group generally performs worse than greedy algorithms. This is especially apparent in the results for the ARM platform. This would be partially the consequence of the misleading, heuristic cost model, which was analyzed in depth in [7]. Regarding our algorithms, single mode performed consistently and by a large margin better than multiple mode. As the one makes up in breadth what the other does in depth, we suspect this is due to the cost impact of tree breadth as described in [7]. It warrants implementing said cost model to our platform to better explain the results. Altogether our algorithms are the second published solution that produces *functionally correct* decoders, together with [10]. As to performance, our algorithms are among the fastest, only slightly beaten for the SPARC platform. Regarding the individual metrics, they produce very similar results for single predictors. We attribute this to the fact that, since only a single bit or predicate is chosen, there is a high likelihood of all four algorithms selecting the same predictor. Multiple predictors obviously offer more space for divergence, with Entropy/Information Gain Ratio performing worse with SPARC but much better with MIPS, compared to Chi-squared and Gini Index.

To conclude, we note the difference in decoding speed between ISAs using propositions (MIPS and ARM) and those doing without them (SPARC), as apparent in the execution time scale. This is a further indication of the impact of ISA complexity on decoder performance. As this trend is expected to continue, the necessity of optimized decoder generation algorithms is further emphasized.

VI. Conclusion

We have conceived and implemented an algorithm that solves the problem of generating instruction decoders for complex and irregular instruction sets. We benchmarked the generated decoders to measure the execution speed. From a functional point of view, only one other published algorithm is currently capable of producing fully automated and functionally correct decoders. From a performance perspective, our algorithms outperform existing decoders. Our future work is aimed at improving the decision function selection algorithm in combination with an accurate cost model.

References

[1] H. Theiling, "Generating decision trees for decoding binaries," in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES)*, jun 2001, pp. 112–120.

[2] T. Ratsiambahotra, H. Cassé, and P. Sainrat, "A versatile generator of instruction set simulators and disassemblers," in *Proceedings of the International Symposium on Performance Evaluation of Computer Telecommunication Systems (SPECTS)*, jul 2009, pp. 65–72.

[3] N. Fournel, L. Michel, and F. Pétrot, "Automated generation of efficient instruction decoders for instruction set simulators," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, nov 2013, pp. 739–746.

[4] K. Okuda and H. Takeyama, "Decision tree generation for decoding irregular instructions," in *Proceedings of the Conference on Design, Automation Test in Europe Conference (DATE)*, mar 2016, pp. 1592–1597.

[5] W. Qin and S. Malik, "Automated synthesis of efficient binary decoders for retargetable software toolkits," in *Proceedings of the Design Automation Conference (DAC)*, jun 2003, pp. 764–769.

[6] L. Tadros, "Functional analysis and performance evaluation of decoder decision tree generation algorithms," in *Proceedings of the 37th International Conference on Modelling and Simulation (ECMS)*, jun 2023, pp. 262–269. [Online]. Available: https://www.scs-europe.net/dlib/dl-index.htm

[7] ——, "A cost model for decoder decision trees," in *Proceedings of the 2020 European Symposium on Software Engineering (ESSE)*. New York, NY, USA: Association for Computing Machinery, nov 2020, pp. 142–147.

[8] H. Wagstaff, "From high level architecture descriptions to fast instruction set simulators," PhD thesis, The University of Edinburgh, nov 2015.

[9] N. Y. Fokina and M. A. Solovev, "Automated generation of machine instruction decoders," vol. 45, pp. 390–397, 2019.

[10] L. Tadros, "Automated generation of decision trees for decoding irregular instruction sets," in *Communications of the 38th International Conference on Modelling and Simulation (ECMS)*, pp. 459–465. [Online]. Available: https://www.scs-europe.net/dlib/dl-index.htm

[11] Z. Cheng, X. Cao, and J. Zhao, "Research on approaches of implementation of instruction set simulator," in *Proceedings of the 2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2021, pp. 1008–1011.

[12] Free Software Foundation, Inc., "GNU binutils," http://www.gnu.org/software/binutils, version 2.38.

[13] F. Bellard, "QEMU. Open Source Processor Emulator," wiki.qemu.org/Main_Page, version 6.2.0.

[14] Y. Klimiankou, "Rapid instruction decoding for IA-32," in *Perspectives of System Informatics*, N. Bjørner, I. Virbitskaite, and A. Voronkov, Eds. Cham: Springer International Publishing, 2019, pp. 1–9.

[15] *The SPARC Architecture Manual*, SPARC International Inc. Std. Version 8, Revision SAV080SI9308, 1992.

[16] *ARM Architecture Reference Manual*, ARM Ltd. Std. ARMv7-A and ARMv7-R edition (C.b), jul 2012.

[17] *MIPS Architecture for Programmers. Volume II-A: The MIPS32 Instruction Set Manual*, Imagination Technologies Ltd. Std. MD00 086, Revision 6.05, may 2016.

[18] L. Rokach and O. Maimon, *Data Mining with Decision Trees. Theory and Applications*. River Edge, NJ: World Scientific Publishing Co., Inc., 2015.

[19] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," *Journal of The Royal Statistical Society Series C-applied Statistics*, vol. 29, pp. 119–127, 1980.

[20] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[21] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[22] ——, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[23] L. Fossati, "trap-gen," https://code.google.com/archive/p/trap-gen/, version 0.4.5.

[24] L.-N. Pouchet, "PolyBench/C," https://web.cs.ucla.edu/~pouchet/software/polybench/, version 4.2.1.