High-Performance Process-in-Memory Architectures Design and Security Analysis

by

Ziyu Wang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan
2024

Doctoral Committee:

Professor Wei Lu, Chair
Professor Todd Austin
Associate Professor Ronald Dreslinski
Associate Professor Hun-Seok Kim

Ziyu Wang
ziwa@umich.edu
ORCID iD: 0000-0001-9534-3134

ACKNOWLEDGEMENTS

"All things pass, a sunrise does not last all morning; All things pass, a cloudburst does not last all day." Now the time for my doctoral years is up. Numerous people have left their traces in my life. And they opened my mind to the material world.

I would express my utmost gratitude to my advisor, Prof. Wei Lu. Wei kindly offered me an opportunity to explore the glamorous research realm when I was about to tread on a brand new path. For years, countless guidance has flown out from him whenever confusion rushes through my head. With his support, the seeds of my ideas can be planted, grew and blossomed. I will always feel fortunate to have worked on several original projects with him.

My committee members, Prof. Todd Austin, Prof. Ronald Dreslinski and Prof. Hun-Seok Kim, gave insightful comments on my thesis. No doubt my dissertation got improved a lot by the profound discussion with them. Earlier than this, what I valued most were the distinguished courses offered by them. My strong foundations are built by the knowledge they given.

It is my pleasure to collaborate with Xiaojian Zhu, Mohammed A. Zidan, Jason K. Eshraghian, Seung Hwan Lee, John Moon, Qiwen Wang, Yuting Wu, Fan-Hsuan Meng, Xinxin Wang, Yongmo Park, Sangmin Yoo, Eric Yeu-Jer Lee, Xiaoyu Zhang, Soohyeon Kim and Xiaofeng Hu. Research during the Covid-19 pandemic was tough. But we made the best of the situation, somehow. I gained most from two of them, and they deserve to be thanked more. Jason consistently provides valuable insights and resources with me, even after his departure. It makes my life much easier. Yuting made wonderful contributions to the project we have done together. How my research and career would be diverted without her contributions, I don't know.

I cannot claim my doctoral study successful without many other people's helping hands. Supreet Jeloka and Brian Cline at Arm Ltd. held monthly discussion on my first project. This exposed me

to the industry at the early stage. ECE program coordinator Kristen Thornton assists a lot when I have trouble in academic. Excellent staffs at ARC, DCO and LNF are always responded even when I ask silly questions. They facilitate the smooth progression of my research. Experience at Apple Inc. enhances my awareness of project planning, propelling my workflow during the graduation process.

I should have more people to thank, but there is not enough space to do so. I sincerely acknowledge all my friends for sharing their time with me, especially when we were crucified by the pandemic. They feed my spirit with thoughts and fill my soul with happiness.

Back through the years, I found the days sticking around Tsinghua profiled me inevitably. It made me initiate my doctoral journey as a journeyman, leaping forth pitfalls. A young man might be too self-ordained to know the unworthiness. But what I learned from those geniuses will silhouette me all the while.

I can never tell how grateful I am to my parents. I take it for granted that there is always a way to get back homeward with their endless love and support.

Before the curtain down, I was not doing nothing but aging. In the end, there will be no other words can retrospect the past few years better than one of Bod Dylan's lyrics,

"Ah, but I was so much older then,

I'm younger than that now."

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

ABSTRACT

The performance of processor-centric von Neumann architectures is greatly hindered by data movement between memory and processor, especially when encountering data-intensive tasks. Memory-centric process-in-memory (PIM) architectures perform computations directly within the memory modules. Hence, the performance and energy penalty associated with data access can be mitigated by minimizing data movement and leveraging high internal bandwidth. In addition to the benefits in performance and energy efficiency, PIM architectures facilitate extensive computing parallelism and scalability, while also provide enhanced security resilience against bus-snoop attacks. PIM architectures have shown their capability in many machine learning applications. Nonetheless, effectively accommodating ultra-large deep neural network (DNN) models, like Transformer, remains an ongoing challenge, and with the continued adoption of PIM architectures, security and vulnerability issues are poised to become looming threats.

This dissertation focuses on high-performance PIM architecture design for data-intensive applications. To facilitate PIM architecture design and security studies, the dissertation first proposes event-driven, cycle-accurate simulators and their implementations for PIM architectures based on dynamic random-access memory (DRAM) and resistive random-access memory (RRAM), along with how these simulators can be used for architecture design.

The PIM-GPT architecture is then introduced, which offers high performance, high energy efficiency and end-to-end acceleration of GPT inference. PIM-GPT leverages DRAM-based PIM solutions to perform multiply-accumulate (MAC) operations on the DRAM chips, working together with an application-specific integrated chip (ASIC) which supports data communication and other necessary arithmetic computations. At the software level, the mapping scheme is designed to maximize data locality and computation parallelism by partitioning a matrix among DRAM channels

and banks to utilize all in-bank computation resources concurrently. Overall, PIM-GPT achieves 41−137×, 631−1074× speedup and 123−383× and 320−602× energy efficiency over GPU and CPU baseline, respectively, on 8 GPT models.

Two security and vulnerability investigations are then conducted on RRAM-based analog PIM architectures. These studies employ a dynamic power trace modeling approach at runtime, enabling efficient power and timing side-channel analysis. The susceptibility of PIM architectures to side-channel attacks is analysed. And the study reveals the possibility of extracting complete DNN model architectural information solely from power trace measurements, without prior DNN knowledge. Furthermore, another potential security vulnerability is identified, wherein an adversary can reconstruct a user's private input data through a power side-channel attack, given proper data acquisition and pre-processing. The study employs a machine learning-based attack approach utilizing a generative adversarial network (GAN) to enhance data reconstruction. Notably, these findings illustrate the effectiveness of specific attack methodologies in extracting DNN model structures and user inputs from analog PIM accelerator power leakage, even in the presence of substantial noise levels. Countermeasures against these side-channel attacks are also discussed.

In light of these security challenges, there is a growing demand for hardware secure systems capable of providing robust solutions for identification, authentication, and protection against counterfeiting and unauthorized modifications. Physical unclonable functions (PUFs) emerge as a valuable technique for hardware root-of-trust. A PUF system built upon fingerprint-like random planar structures is developed, demonstrating compatibility with the back-end-of-line (BEOL) process and presenting promising potential as a hardware security primitive in the IoT industry.

In the end, guiding principles and proposals for future work are deliberated, focusing on three key aspects: 1) hardware modeling and simulation of emerging PIM architectures; 2) hardware/software co-optimization for Transformer models; and 3) security and vulnerabilities in neuromorphic computing systems.

CHAPTER 1

Introduction

1.1 Background

Modern computing systems, stemming from the von Neumann architecture, are processor-centric architectures. As illustrated in Figure 1.1(a), the conventional CPU architecture emphasizes data processing within the arithmetic logic units (ALU). Consequently, data must traverse the memory hierarchy to reach the processor. However, the transmission of data through multi-level caches introduces substantial latency. Moreover, external memory access incurs energy costs approximately two orders of magnitude higher than those associated with typical ALU processing tasks, such as floating-point multiplication [1][2]. The latency and energy costs associated with external memory access, commonly denoted as the von Neumann bottleneck, significantly curtail the performance of contemporary computing systems, particularly when handling extensive data [3][4].

In the era of big data, data-intensive applications such as machine learning [5], graph processing [6][7][8] and genome sequence analysis [9] require more efficient data processing. Effectively managing data emerges as the paramount challenge [10]. Modern computing architectures face several intrinsic challenges that impede their effectiveness. For example, their fundamental design prioritizes data storage and movement, lacking the nuanced optimization necessary for efficient data consumption. More specifically, a significant portion of the chip area is allocated to accommodate data through the use of cache or scratchpad memory, aiming to minimize data access latency.

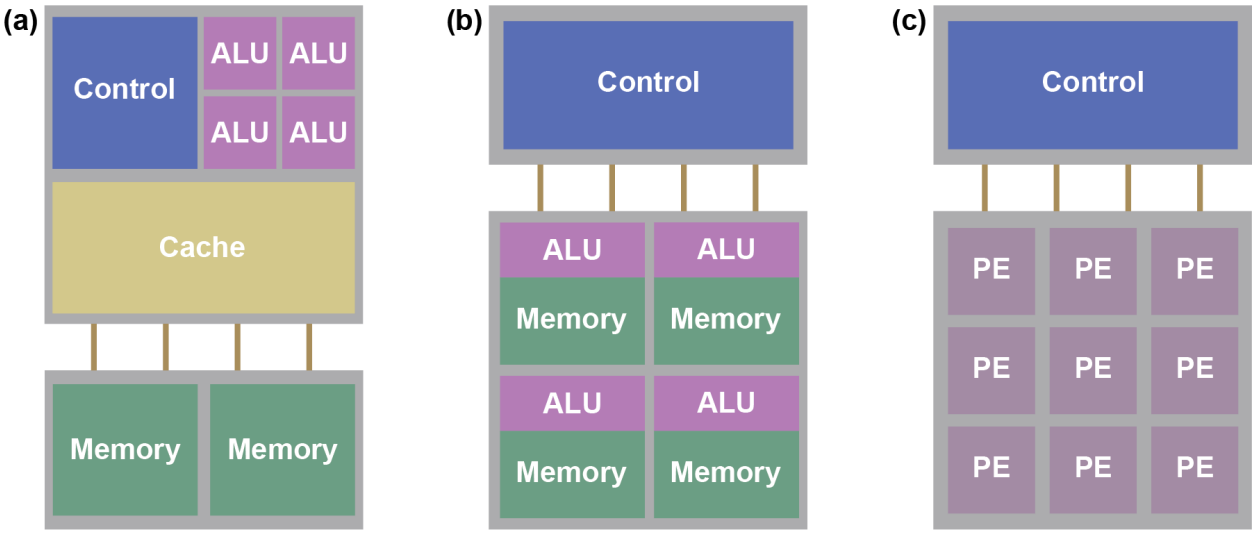


Figure 1.1: (a) CPU based on conventional von Neumann architecture. (b) Process-near-memory (PNM) architecture. (c) Process-using-memory (PUM) architecture.

But the memory access latency of individual piece of data can not be optimized. Moreover, the architectures struggle to harness the diverse applications and inherent properties of data effectively. The current general purpose processors often fall short in capitalizing on the rich and varied nature of data and applications, hindering their ability to cater to the specific needs of different applications and exploit the unique characteristics of distinct datasets. Consequently, the future trajectory lies in the redesign of architectures to unlock their full potential in handling the complexities of modern data processing challenges.

To resolve the aforementioned challenges, computing architectures should pivot towards a data-centric and application-driven design. Bringing processing elements (PE) in proximity to the location where data resides or is generated emerges as a straightforward and potent strategy. This principle underlies the motivation behind process-in-memory (PIM) architectures.

1.2 Process-in-Memory (PIM) Architectures

PIM architectures are categorized as memory-centric computing paradigms, with the primary goal of conducting computation within or in close proximity to data. Therefore, it can potentially

alleviate the performance and energy penalties associated with data movement. The root of PIM can be traced back to the 1960s, when the concept of logic-in-memory was first proposed [11][12]. In recent years, PIM has gained prominence in both industry and academia. This upswing is distinctly propelled by the burgeoning need for handling data-intensive tasks such as artificial intelligence and machine learning, a demand fueled by the exponential growth in data-driven applications. Simultaneously, the ascent of PIM has been bolstered by significant strides in memory technology. Besides implementing computation capability into the mainstream DRAM chips [2][13][14][15][16], other emerging memory technologies also pave brand new avenues for high performance and low power computing architectures that directly utilize the memory devices to perform computation [17][18][19].

PIM architectures can be fundamentally divided into process-near-memory (PNM) and process-using-memory (PUM), as shown in Figure 1.1(b) and (c). The two approaches can also be combined to enhance overall performance [20]. The architecture of PNM does not deviate significantly from traditional processor-centric systems. As shown in Figure 1.1(b), PNM architectures still have distinct processing units and memory arrays. However, the strategic placement of processing units closer to memory significantly mitigates the cost associated with data movement. Moreover, near-memory logic can take advantage of the high internal bandwidth, surpassing that of off-chip memory interfaces. As for DRAM chip, such a near-memory logic can be placed near memory banks [13][14], close to memory sub-arrays [21][22] or on the logic layers in 3D-stacked dies [23]. Besides DRAM technologies, PNM can also be employed for caches [24] and storage devices [25]. This versatility underscores the adaptability of PNM architectures across a spectrum of memory-intensive components in computing systems.

PUM harnesses the analog behaviors inherent in device operations. Specifically, some analog devices exhibit stateful transition behaviors that provide a foundation for computation directly within memory devices [26]. In the PUM paradigm, computation takes place within the memory cells without the need for additional peripheral arithmetic circuitry, as shown in Figure 1.1(c). This fundamentally distinguishes the computing approach of PUM from that of modern computer archi-

tectures. Despite the precision limitations in analog computing when compared to floating-point operations, the performance of certain tasks, notably machine learning inference, can potentially tolerate the reduced precision [27]. Analog PUM computations have been widely supported by emerging memory technologies, which will be elaborated more in the Section 1.3. PUM has also been deployed with conventional memory types, including SRAM [28][29], DRAM [30] and Flash [31] to accelerate machine learning applications. Since PUM architectures rely more on internal behaviors of memory devices, the design and optimization of the devices become critical.

PIM-enabled memory-centric computing architectures have four primary advantages. First of all, it can essentially reduce the data movement between the processor and memory, ensuring that all data are consumed at storage location. This mitigates performance bottlenecks arising from processor-memory speed mismatches, resulting in substantial improvements. Secondly, PIM architectures exhibit lower latency and energy requirements for accessing individual data since the PEs are much closer to data. Many recent works have shown that PIM architectures can improve both performance and energy efficient by more than one order of magnitude [6][20][32]. Thirdly, PIM architectures facilitate extensive parallel computing. By integrating PEs at bank or array level, parallel operation of all submodules can be achieved with the same instruction, concurrently reducing instruction overhead. Lastly, PIM architectures are considered more secure compared to processor-centric architectures. Specifically, they are more resilient against bus-snoop attacks [33][34]. Co-locating memory and processing eliminates data movement through the system bus, making it more challenging for malicious users to compromise the security of these hardware architectures. The limited memory access of stationary weights adds an additional layer of security, enhancing the robustness of PIM architectures.

Memory-centric PIM architectures can accelerate data-intensive tasks such as large language model (LLM), convolution neural network (CNN) for computer vision, graph processing, cryptography computing and genome analysis [6][7][9][20][32][35]. Moreover, emerging devices offers a novel approach to utilize their intrinsic dynamic state behaviors [36], which introduces an avenue for new computation paradigms such as neuromorphic computing, spiking neural networks and

reservoir computing [37][38][39].

1.3 Devices and Micorarchitectures for PIM

1.3.1 DRAM-based PIM Architectures

SRAM and DRAM have been considered for PIM architectures due to their maturity. By redesigning cache peripheral circuitry and the instruction set architecture (ISA), instructions can be prompted in-cache [28][29]. Using dual port SRAM decouples read and write, which makes it possible to utilize intrinsic analog behavior of SRAM cells to accelerate multiply-accumulation (MAC) operation in-situ [40][41]. However, the density of SRAM is restricted by its 6T (transistor) cell design (which can add up to 8T or 10T to enable more complex functions), so the capacity of SRAM-based PIM is bounded by the area constrains and poor scalability.

As a comparison, the capacity of DRAM chips is much larger, ranging from several GB to tens of GB. State-of-the-art GPU NVIDIA A100 can be equipped with 80 GB high-bandwidth memory (HBM) [42][43][44]. Moreover, multiple DRAM channels and chips can be easily scaled at the board level. Enhanced inter-DRAM communication can further improve its scalability [45].

A DRAM chip composes of multiple banks. Figure 1.2(a) shows the DRAM bank architecture, with inset shows a single 1T1C DRAM cell. The capacitor cell represents the binary value of 0 or 1 by the absence or presence of charge. The transistor is driven by word-line voltage to access the cell. Bit-lines of a bank are connected to the row-buffer. After opening a bank row, the entire row's data are read and temporarily stored in the row-buffer. Subsequently, the target data is accessed using the column address. Read and write operations of a certain DRAM cell are shown in Figure 1.2(b). To start with, all bit-lines are maintained at a voltage-level of $\frac{1}{2}V_{DD}$. An activation command is received along with the row address, the corresponding word-line voltage will be raised to V_{PP} from 0. The data from the entire row will be sensed by the row-buffer. After that, the memory controller will issue a read or write command, along with the column address to fetch the corresponding data in the row-buffer. If the next request tries to access the same row, it can directly access the row

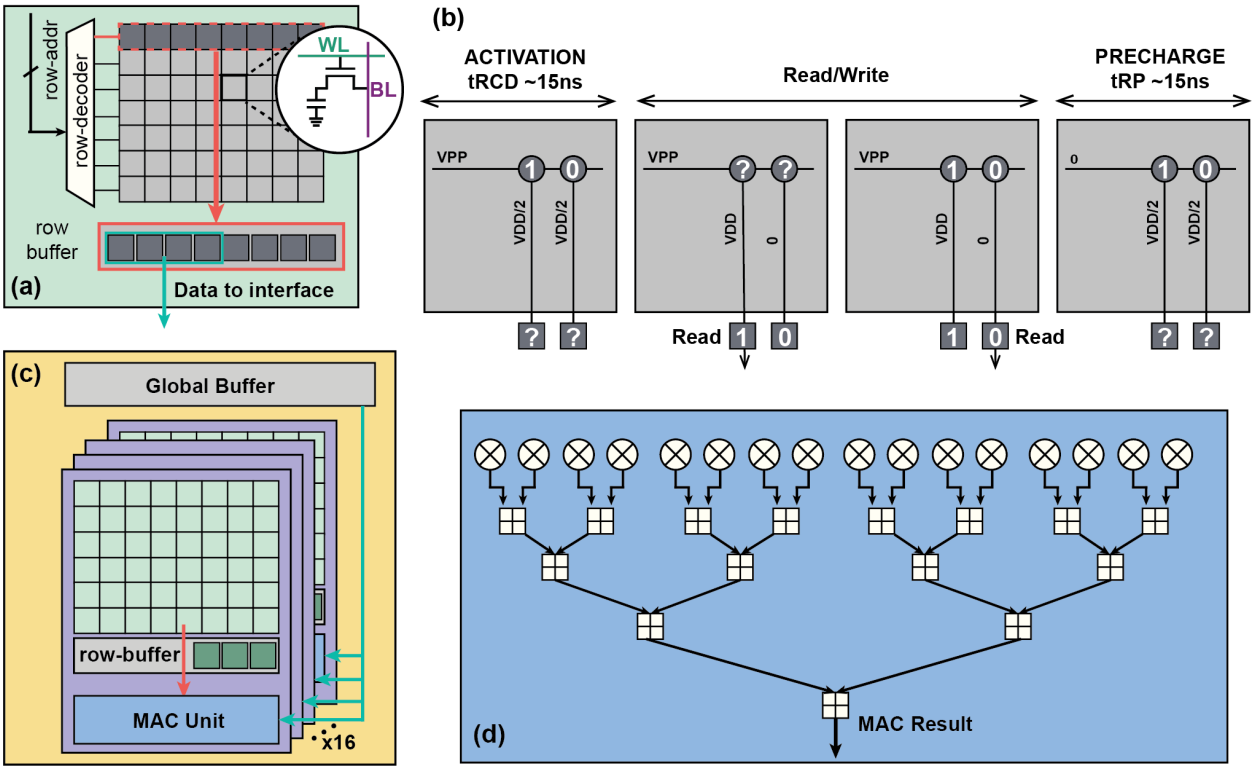


Figure 1.2: (a) DRAM bank architecture, inset: 1T1C single cell. (b) Schematic of DRAM read/write operations. (c) DRAM channel with arithmetic logic. (b) Multipliers and an adder tree for MAC operation.

buffer without activate the row again. By doing so, the latency can be greatly reduced. In the end, if a new row needs to be activated, the memory controller must precharge the current row. The word-line voltage will be lowered to 0 and bit-line voltage will be driven to $\frac{1}{2}V_{DD}$. The latency of DRAM operation must follow DRAM timing constraints [46].

To enable PUM in DRAM, extensive modifications to the subarray microarchitecture are needed [30][47][48]. And only a limited instruction set can be supported. Therefore, adopting PNM to DRAM is a better option, which adds computation logic near DRAM bank and takes advantages of high internal bandwidth. Adding computation logic will sacrifice the memory capacity since the die area cannot be expanded. Moreover, the DRAM fabrication process is highly constrained for logic integration. It only contains three metal layers which severely limits the complexity of the circuit, and the transistors are 3× slower than those in logic chips at the same node [5]. Hence, adding complex logic components into DRAM is not practical.

Due to the above-mentioned technical difficulties, PIM architectures are not integrated into memory products until recently. UPMEM is the first reported PIM product using standard dual in-line memory module (DIMM) modules, with a large number of processors combined with DRAM chips [16][49]. However, the throughput per processing unit (PU) is 4 GOPS and it only supports INT8 data type. DRAM vendors including Samsung [2][14] and SK Hynix[13][15] have recently announced DRAM-based PIM technologies. Samsung's PIM architecture is based on HBM2, which offers high bandwidth of 307.2 GBps to tackle data-intensive tasks. The design integrates PIM dies on a buffer die through TSV. Inside each PIM die, a processing unit (PU) is shared by two banks, operating at 9.6GFLOPS per PU. However, the high cost of HBM limits the application to server level. SK Hynix's GDDR6-based PIM prototype, Accelerator-in-Memory (AiM), uses the standard GDDR6 interface. AiM supports VMM with a high throughput of 32GFLOPS per PU.

Figure 1.2(c) shows a representative DRAM-based PIM microarchitecture design. A global buffer stores intermediate data and broadcast to every bank. MAC units are integrated at bank-level to maximize parallel computing. MAC units can be implemented by multipliers and an adder tree, as shown in Figure 1.2(d).

1.3.2 Emerging Memory-based PIM Architectures

Besides standard memory devices that store information using charge, emerging memory technologies typically store data using other mechanisms. Some of these emerging devices are optimized for non-volatile storage and analog switching behavior, while others with dynamic properties can mimic synaptic functions [37][50][51][52]. Figure 1.3 illustrates representative emerging memory devices. Resistive random-access memory (RRAM) operates based on internal ion redistribution. Given the different type of filament, based on either oxygen vacancy [53][54] or active metals such as Ag [55][56], RRAM can be categorized into valence change memory (VCM) and conductive bridge random-access memory (CBRAM), in Figure 1.3(a) and (b) respectively. The gradual formation and rupture of the filament can represent analog conductance states [57]. Besides memory application, the nanoscale filament structure offers a platform for fundamental chemical and

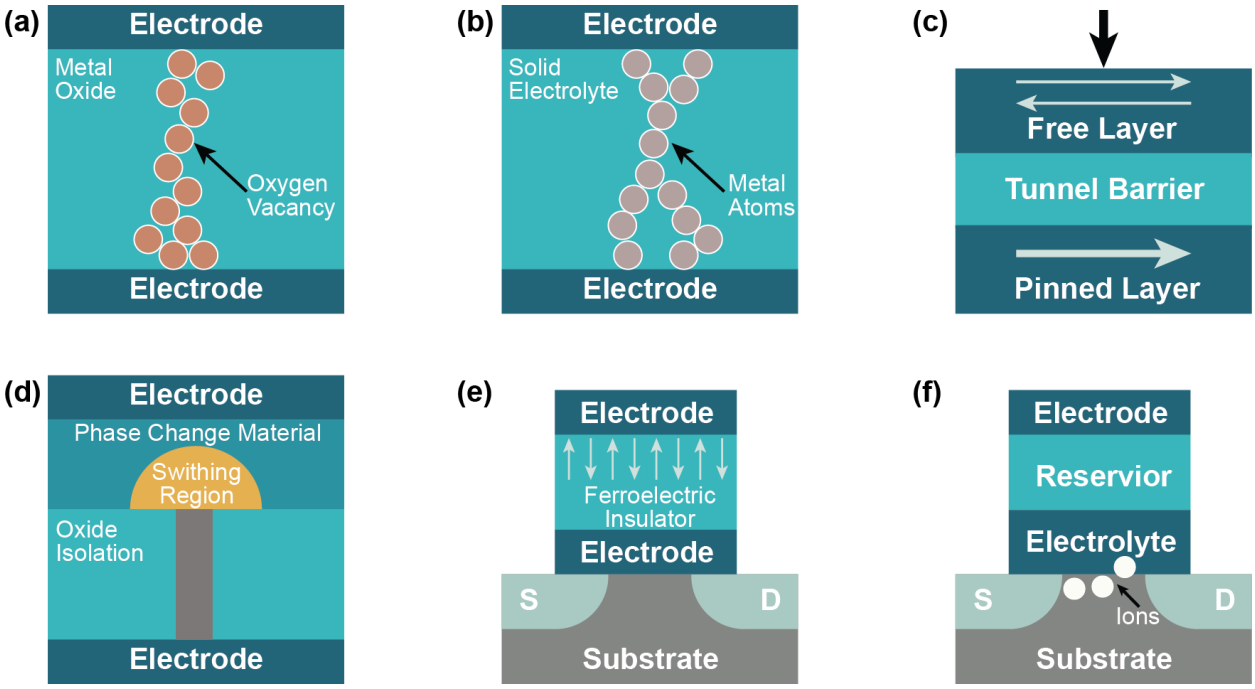


Figure 1.3: Illustration of emerging memory devices. (a) RRAM, (b) CBRAM, (c) STT-MRAM, (d) PCM, (e) FeFET, (f) ECRAM.

physical effect studies [58][59][60][61]. Spin-transfer-torque magnetic random-access memory (STT-MRAM) stores information using the magnetization direction between a free layer and a pinned layer [62], and the parallel/anti-parallel directions of these layers can be modified by spin current injection. Phase change memory (PCM) operates on the conversion of chalcogenide-based material between crystalline and amorphous forms through joule heating [63][64]. It can also achieve analog conductance states. Ferroelectric field-effect transistors (FeFETs) have a structure similar to that of conventional transistors but utilize a ferroelectric insulator layer as the dielectric layer [65][66]. The application of a gate voltage induces polarization switching in the ferroelectric layer, modulating the conductance of the channel in FeFETs. Electrochemical random-access memory (ECRAM) can achieve more stable analog conductance. Under the electric field induced by the applied gate voltage, ionic transporting between the electrolyte and the channel modulates the channel conductance [67][68].

Among these emerging memory technologies, RRAM is one of the most promising candidates for its low power, high speed, non-volatile storage, superior scalability and CMOS-compatible

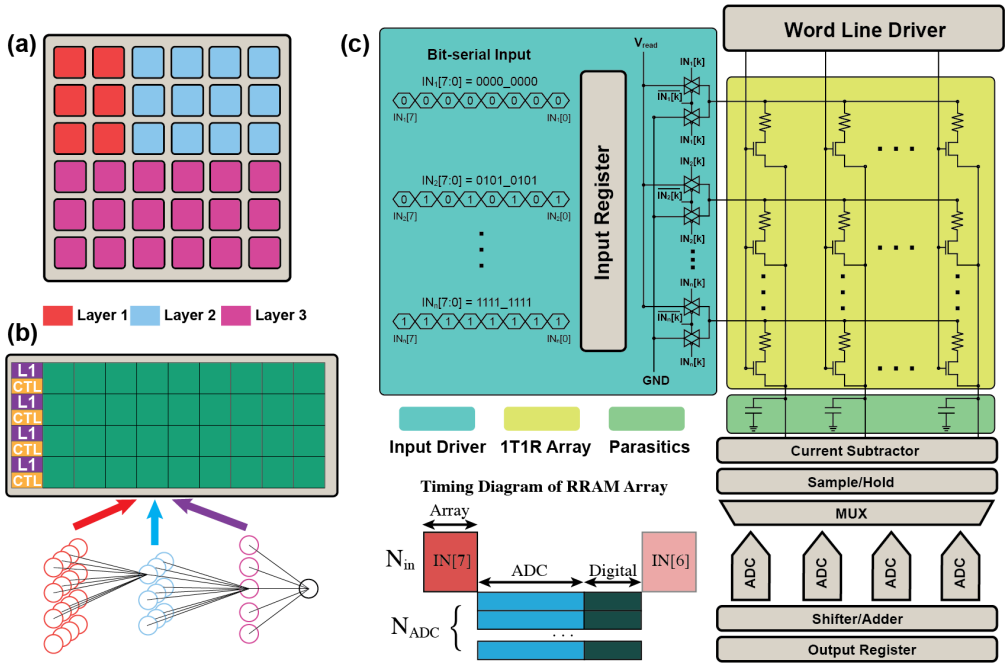


Figure 1.4: (a) Spatial-multiplex RRAM-based PIM architecture. (b) Timing-multiplex GPU architecture. (c) RRAM array with peripheral circuitry.

process [69][70][71]. RRAM, also referred to as the memristor, was first proposed theoretically in 1971 [72]. Memristor was proven to support synaptic functions in 2010 and broadened the way for neuromorphic computing [73]. Later in 2015, crossbar array formed by memristors was experimentally used for neural network operations [74]. RRAM will be one of the main focuses of the thesis.

Figure 1.4 shows a representative RRAM-based PIM microarchitecture, where mixed-signal circuits are integrated within RRAM crossbar macros, and analog computation is achieved through bit-line current summation or charge accumulation. Co-locating memory and processing in a tiled architecture eliminates data movement between memory and processor, as shown in Figure 1.4(a). In this case, different layers can be processed concurrently in space-multiplexed fashion. In contrast, conventional deep neural network (DNN) accelerators, such as GPUs, use single-instruction-multiple-threads (SIMT) execution and must therefore time multiplex operations that take place across different DNN layers, as depicted in Figure 1.4(b).

Analog PIM architectures offer significant advantages in throughput and power efficiency by

minimizing data movement, and offer a high degree of parallelism at runtime with respect to MAC operations [75][76]. Figure 1.4(c) shows a detailed schematic of an analog RRAM crossbar array-based PIM. The analog RRAM PIM can perform vector-matrix multiplication (VMM) directly in a single step based on current summation: Ohm's law is used for multiplication and Kirchhoff's current law is used for accumulation. More specifically, the input vectors are encoded as voltage pulses and the entries of matrices are mapped as RRAM device conductance values. The outputs of the VMM are returned as bit-line currents, subsequently sampled by peripheral read-out circuitry, and converted to binary digital values using analog-to-digital converters (ADC) for further down-stream communication and processing. As the weight precision is often higher than the device precision, typically a single weight value is mapped across multiple RRAM cells, and the VMM result is reconstructed using digital shifter and adder circuits [77].

1.4 Security and Vulnerability of PIM Architectures

Although DNN models are implemented on a wide range of platforms, from datacenter GPUs to edge devices, the importance of data security cannot be overstated. Attacks on DNN data used in critical applications such as medical diagnosis, autonomous driving, and financial transactions can compromise the user privacy as well as proprietary algorithm information [78]. The main areas of concern in DNN security include model extraction, adversarial attacks, and privacy breaches [78] [79] [80]. Model extraction involves extracting the DNN architecture and reconstruct weights to reproduce its functionality, which can be used to breach the intellectual property of DNN design [79]. Adversarial attacks involve manipulating the input data deliberately to make the modification invisible to human but causing misclassification or other unexpected behavior to the DNN model [80]. Privacy breaches involve reconstructing sensitive private input or training data by an attacker [78] [81].

A typical attack of the DNN hardware is through side-channel, as schematically shown in Figure 1.5. Side-channel attack analysis is based on physical phenomena during execution, such

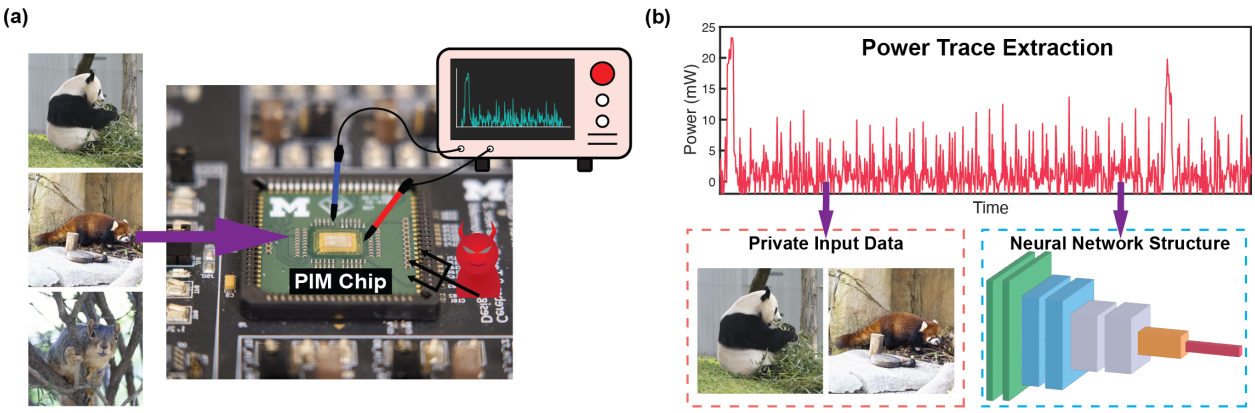


Figure 1.5: Schematics of side-channel attack. (a) Side-channel leakage measurement of PIM chip. (b) Power side-channel leakage with potential sensitive data.

as electromagnetic emanation, power dissipation and timing, as well as mathematical analysis. By carefully measuring and analyzing the power dissipation of the chip, attackers may be able to reverse engineer sensitive data or architectural information [82]. Figure 1.5(b) shows privacy breach and model extraction attack by analyzing the power traces of PIM system at runtime.

While the above-mentioned security attacks on DNN hardware have been extensively evaluated on systems such as GPUs, CPUs and FPGAs [79] [81] [83] [84] [85] [86], security and vulnerability analysis of analog PIM accelerators is largely lacking. While PIM macros reduce the number of possible attack vectors, the risk of side-channel attack remains a looming threat, as information leaks may still occur via power profiling and electromagnetic emanations. Timing analysis of thread-level execution may also offer an unintended window into architectural insights. Like the developments of RowHammer [87] changes the whole DRAM industry, studying the vulnerability in PIM systems will empower the security considerations in PIM architecture design.

1.5 Organization of the Thesis

This thesis targets on high-performance PIM architecture design, as well as security and vulnerability analysis of PIM systems. PIM designs based on two representative memory technologies, mainstream DRAM and emerging RRAM, are studied in this thesis. The thesis will start with PIM

architecture modeling and high-performance PIM architecture design. A DRAM-based PIM architecture will then be introduced for large Transformer model inference acceleration. For emerging memory technology such as RRAM, two side-channel attack and reverse engineering approaches, model extraction attack and private input breach, will be introduced, followed by discussing potential countermeasures. To further secure the chips, hardware security primitive physical unclonable functions (PUF) will also be discussed.

The content of each chapter is summarized below.

Chapter 2 discusses how to design and implement event-driven, clock-cycle-accurate simulators for PIM architectures based on DRAM and RRAM. The simulators offer fast architecture evaluation along with accurate latency results for different benchmark analysis. An example of designing RRAM-based analog PIM architecture guided by simulation will be elaborated.

Chapter 3 proposes a DRAM-based PIM design, PIM-GPT, that aims to achieve high throughput, high energy efficiency and end-to-end acceleration of GPT inference. PIM-GPT leverages DRAM-based PIM solutions to perform MAC operations on the DRAM chips, along with a compact application-specific integrated chip (ASIC) to support data communication and necessary arithmetic computations.

Chapter 4 proposes a side-channel attack methodology on RRAM-based PIM architectures. It shows the feasibility to extract model architectural information from power trace measurements without any prior knowledge of the neural network. Practical side-channel measurement methodologies are also discussed. Potential countermeasures for building secure PIM systems are studied.

Chapter 5 identifies a potential security vulnerability wherein an adversary can reconstruct the user's private input data from a power side-channel attack, under proper data acquisition and pre-processing conditions even without knowledge of the DNN model. A machine learning-based attack approach is further demonstrated using a generative adversarial network (GAN) to enhance the data reconstruction.

Chapter 6 discusses a PUF implementation based on a fingerprint-like random planar structure evolved from binary polymer mixture phase separation. The fingerprint PUF is compatible with

back-end-of-line (BEOL) process and provides great potential for hardware security primitive in IoT industry.

Chapter 7 summarizes the thesis and provides outlooks for future research directions. One future project covers defining, architecting, designing, implementing and deploying bit-accurate, cycle-accurate and transaction level simulators for PIM architectures based on DRAM, RRAM and other memory technologies, such as 3D NAND Flash. With more comprehensive simulators, hardware-software co-optimization of PIM architectures can be further studied to support Transformer-type of models. As for security perspective, outlook for security in neuromorphic systems will be discussed along with the security in PIM architectures.

CHAPTER 2

Event-Driven Cycle-Accurate Simulators for PIM Architecture Study

2.1 Background and Motivation

Recent years have witnessed rapid developments of PIM architecture prototypes from both academia and industry. These PIM architectures are based on multiple memory technologies, such as the mainstream DRAM and emerging non-volatile memory including RRAM, and are proven effective in a wide range of machine learning applications. To facilitate these new architectures, there is a growing need of accurate PIM simulators. Simulators play a crucial role in PIM architecture research by providing a flexible, cost-effective, and scalable platform for experimentation, analysis, and optimization of diverse hardware architecture designs.

Several simulators for PIM architectures based on DRAM [88][89][90] and RRAM [91][92][93] have been proposed. They are developed for multiple DRAM standards with insightful information in circuit, architecture and system. And some of these frameworks can integrated with widely-known simulators, such as gem5 [94] and Ramulator [95]. However, existing PIM simulators lack customization for machine learning applications, making them less flexible and efficient for evaluating the performance of various machine learning models, especially when integrating advanced model mapping techniques. Moreover, DRAM-based PIM cannot accelerate non-linear functions efficiently. Optimal acceleration of these functions necessitates lightweight FPGAs or ASICs rather than CPUs or GPUs typically used in the PIM systems. Consequently, the system-level

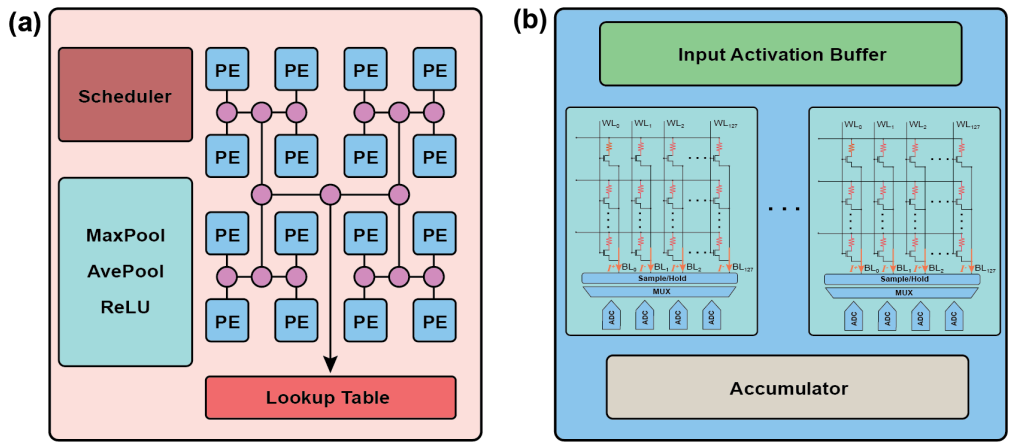


Figure 2.1: (a) RRAM-based PIM architecture design. (b) Processing element organization.

simulators must contend with data transfer and computation occurring on separate chips to fulfill these requirements. On the other hand, current simulators designed for PIM architectures utilizing non-volatile memory, such as RRAM, have different focuses. Given the relatively immature status of these technologies compared to DRAM, some simulators provide circuits-level modeling tools to benchmark the overall hardware performance, such as area, power, energy, and latency. While others target on gauging the impact of device non-ideality on machine learning inference accuracy. However, transaction-level modeling is frequently overlooked in these simulations.

Given the above limitations in existing PIM simulators, there is a strong need in developing a comprehensive, full-system, and clock-cycle-accurate transaction-level PIM simulator specifically tailored for machine learning applications. This chapter focuses on the development of a PIM simulator based on RRAM and DRAM. The simulation methodology presented herein serves as the foundational framework for the subsequent chapters, spanning from Chapter 3 to Chapter 5. The simulators employed in this research are event-driven simulators. Each hardware component is extracted as a state machine, with functions for requesting, executing and sending events. All events will be scheduled for execution on the global time stamp. Hence, the simulator will provide detailed transaction-level information as well as an accurate latency results with various reconfigurable hardware settings.

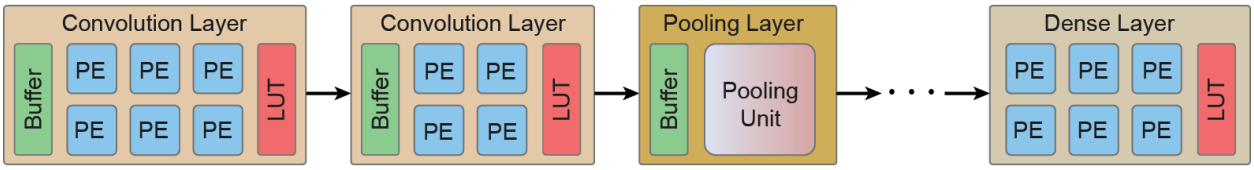


Figure 2.2: Chained data flow architecture.

2.2 PIM Simulator Design

2.2.1 Simulator for RRAM-based PIM Architecture

Figure 2.1(a) shows the schematic of an RRAM-based PIM architecture designed to support the end-to-end machine learning applications. The system contains an instruction scheduler, procesesing elements (PEs) for MAC operations, Network-on-Chip (NoC) for data forwarding and logic units for other arithmetic functions in DNNs. A specific logic unit is responsible for simple logic functions such as pooling and ReLU. As for non-linear functions such as tanh, softmax, the MAC results will be decoded into the indices of the lookup table (LUT), and return the results with the given precision. A PE in Figure 2.1(b) contains an SRAM buffer to store input activation functions, and RRAM crossbar arrays having the same hardware implementation as in Figure 1.4(c). For large neural network layers, the weight matrix needs to be partitioned and split to multiple crossbar arrays, and the accumulator will accumulate and concatenate partial results for downstream computations.

RRAM-based analog PIM systems employ data flow architectures for DNN inference. The pre-trained model weights are flattened and mapped on RRAM tiles. From the DNN layers perspective, PEs, along with other components are allocated spatially to compose different DNN layers, and these layers connect through the NoC based on the model configuration, as shown in Figure 2.2. PEs accelerate the VMM operation in the convolution and dense layers. The partial results from each tile will be accumulated and executed with an activation function before forwarding to the next layer. This defines how hardware resources are utilized with a given machine learning model.

A special consideration of the convolution layer is data reuse. As a pixel will be used for multiple

convolution windows, it should be kept in the buffer until fully utilized. As the input feature map of convolution layer can be large in the first few layers, and be with deeper channels in the later layers, holding all data of the input feature map in the buffer will be area-consuming, power hungry and impractical. An efficient way to solve this problem is to store the data required for a convolution execution only. As the convolution is executed in row-major fashion, the buffer is implemented with a ring structure to fetch the input to its entry with a head and a tail pointer to indicate the valid data. Figure 2.3 shows how the input feature maps of a 2D image are stored in the SRAM buffer. Figure 2.3(a) shows the buffer only stores the data required for a convolution window. Once the buffer has the sufficient data for a convolution execution, the data will be forwarded to the RRAM crossbar array. In the meantime, the tail pointer will also move based on stride configuration, discarding the consumed data. And the new data in the feature map can be filled into the head of the buffer. Once the convolution window reaches the end of the row, the tail pointer will discard all data of that row. Take Figure 2.3(a) as an example, where the convolution kernel size is three, three pixel data will be discard before the convolution window sweeping the next row. Similarly, when the convolution window reaches the end of the feature map, the tail pointer will discard all data in the current feature map. Compared to storing all data in the input feature map showing in Figure 2.3(b), dynamic buffer allocating can greatly save the required hardware resources and better handle pipeline execution. On the other hand, since there is data size and execution latency mismatch among all layers, the buffer size should be carefully evaluated to balance latency and size requirements.

The simulator is implemented in C++ and executed following top-down procedure, i.e. from neural network architecture to allocated hardware resources. All layers, including convolution layers, dense layers and pooling layers, are inherited from an abstract layer class to control sub-modules. The layers form a chained data flow architecture, as shown in Figure 2.2. Data are always forwarded from previous layers to later ones. Each layer, as well as each sub-module embedded in the layers, shares similar data flow. The member functions of all modules can be divided into four categories, data requesting, data forwarding, execution and state update. As shown in Figure

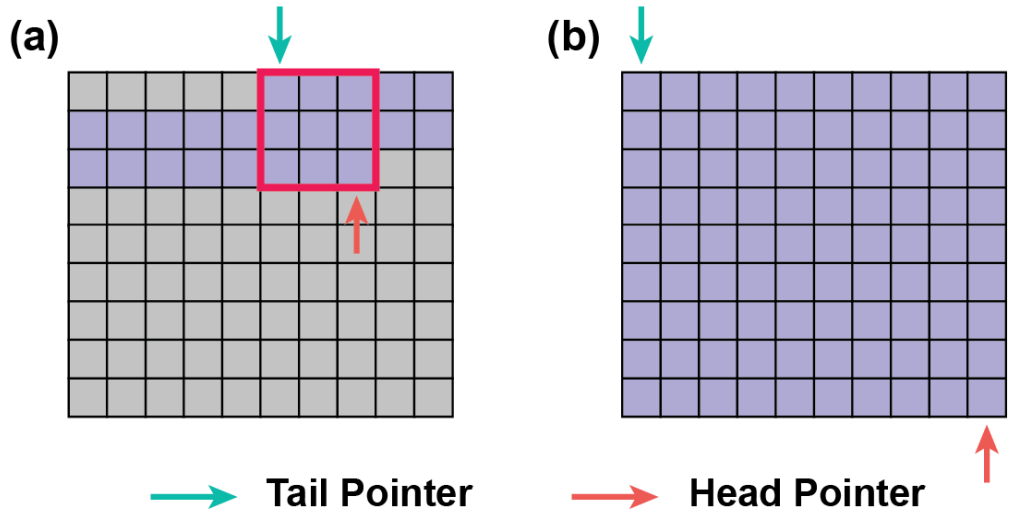


Figure 2.3: Buffer size configurations. (a) Minimum and (b) maximum buffer size for the input feature map.

2.4(a), each layer or module can only request data from the adjacent front one, and the layer can send data only when its data is ready and received a request. Once achieving such a handshake, data will be send and the associated event will be recorded. The simulator is driven by the event communication at clock-level precision. All events of each module will be recorded for statistical analysis. Besides the transaction-level events, data computation will also be supported in each sub-moduel, such as PEs for VMM and LUT for non-linear functions. Meanwhile, the simulator supports multiple system configurations at the DNN model level, the architecture level as well as the circuit level. Table 2.1 summarizes the reconfigurable hardware settings.

At runtime, the next layer will be ready to request data when the bus is not busy and the buffer or tile input register is at the `Ready` state. Then it will request data from the previous layer. As shown in Figure 2.4(a), only when the previous layer has the output data, meaning computation event has completed, data will be forwarded and an event will be scheduled in the next layer based the event firing time. The scheduled event ready time is based on the current clock and the latency of the execution time. A similar data requesting scheme is employed inside each layer for the sub-modules. At every clock cycle, each layer and module will check the event time and the global clock. If the event execution is done at this clock cycle, there will be a state change.

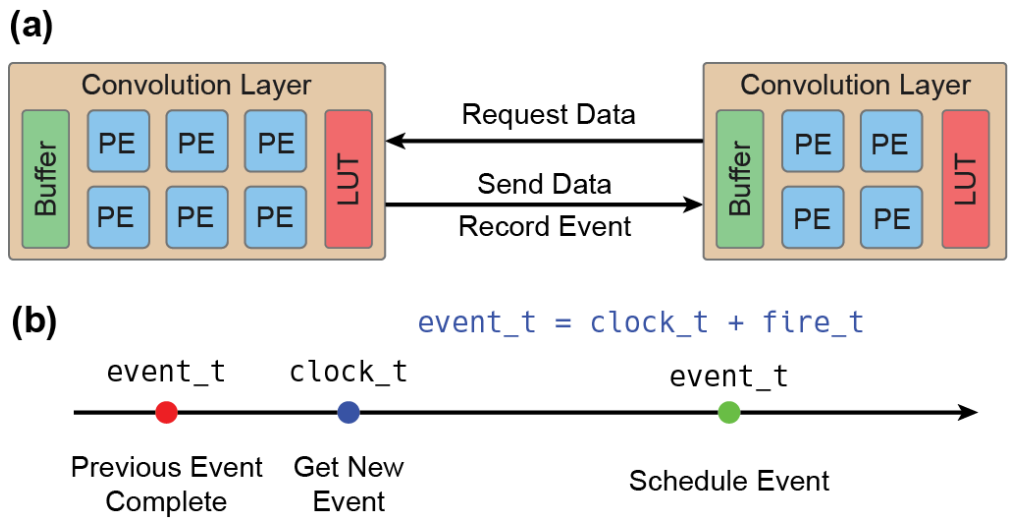


Figure 2.4: (a) Event passing between adjacent hardware components. (b) Event scheduling on the global time stamp.

Table 2.1: Default system configuration

Module	Bus Width	Buffer Size	LUT#	ADC#
Default Configuration	256	Just-right	2	4
Module	ADC Type	Array Size	Model	Device
Default Configuration	SAR ADC	256×64	8 bit	4 bit

2.2.2 Simulator for DRAM-based PIM Architecture

To evaluate the DRAM-based PIM system performance, a transaction-level simulator is developed for modeling. Since DRAM-based PIM runs digital computation, the simulator only models transaction-level information rather than bit accurate computing for fast system modeling. The machine learning instructions and computation flows are deterministic, i.e. computation blocks follow a certain sequence and the later one will only start after previous execution is completed.

For large machine learning model acceleration, multiple DRAM channels are required to compose a PIM fleet. Given the DRAM organization, the PIM fleet is organized as a tree structure, as shown in Figure 2.5. The PIM hierarchy is at levels of fleet, channel and bank. The PIM fleet node refers to the entire PIM portion that contains 8 channels as child nodes. Each channel consists of 16 banks as leaves. Every bank has a memory array and PE, which are stateless. The state

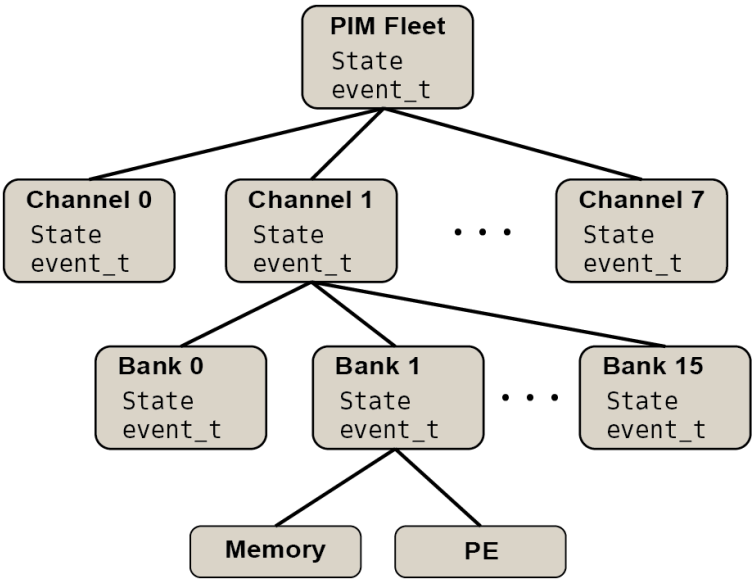


Figure 2.5: Tree structure of DRAM-based PIM fleet, each node is a state machine.

update will traverse down the tree from the root to the leaves. Each node maintains its own states and state-transition sequences. The same organization hierarchy can be found in other memory simulators, such as Ramulator [95]. Scaling can be evaluated by attaching more channel nodes to the fleet node, as well as equipping banks with different processing hardware configuration.

However, the DRAM fabrication process is highly constrained. It only contains three metal layers that severely limit the complexity of the circuit, and the transistors are 3× slower than those in logic chips at the same node [16][49]. As a result, only limited logic and buffers can be integrated on DRAM-based PIM. Therefore, PIM architecture should collaborate with other components, such as a host, to achieve end-to-end machine learning acceleration, as shown in Figure 2.6(a).

As shown in the flowchart in Figure 2.6(b), the simulator takes both machine learning model and DRAM configurations as inputs to determine the model mapping and instruction compilation. Both DRAM and the host are treated as state machines. At every clock cycle, the simulator checks the status of the host and the PIM. The next instruction in the specific instruction stream will be fetched by the controller only when the current instruction is completely consumed, that is, both the host and DRAM are in `Idle` state and no command is pending in the queue. The new instruction is compiled into a sequence of commands based on the mapping and appended to the command queue.

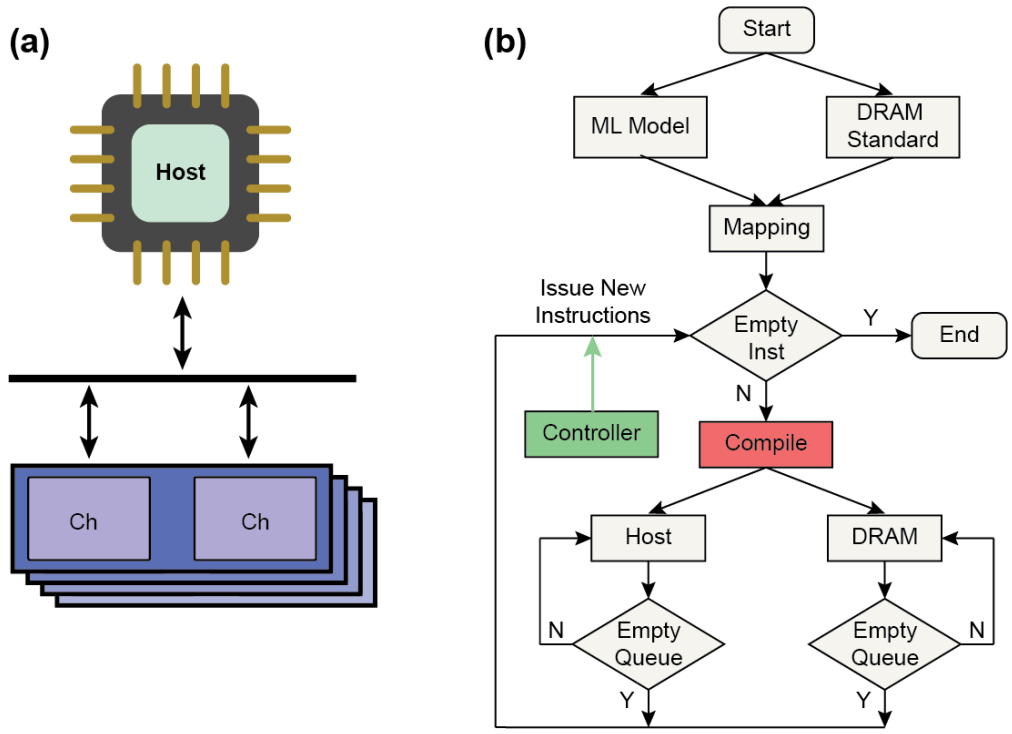


Figure 2.6: (a) System of DRAM-based PIM, containing DRAM channels and a host. (b) Flowchart of PIM system for machine learning acceleration.

The command packets contain information of the operation type, input and output vector lengths, and DRAM address. The compilation considers the hardware resource details. For example, when the size of the input vector exceeds the maximal length of supported VMM operation, the vector will be sliced into smaller chunks and fed to the memory sequentially, followed by adding accumulation operations to the host command queue.

The state machines of a DRAM-based PIM fleet and the host are shown in Figure 2.7. Two sequences are defined for the PIM fleet node for VMM and write operations, respectively. They both start with `Idle` → `Receive` → `Execute`. The VMM results from PIM should be transferred back to the host for arithmetic computation or inter-PIM communication. The state changes from `Execute` → `Transmit` → `Idle`. For write, the fleet node directly switches from `Execute` → `Idle`, since no data needs to be sent to the host. While the host supports multiple functions rather than VMM, the input data need to be stored in the local storage, followed by forwarding to required processing engine. Hence, the state machine will switch to `Idle` before `Execute`. Similarly, as

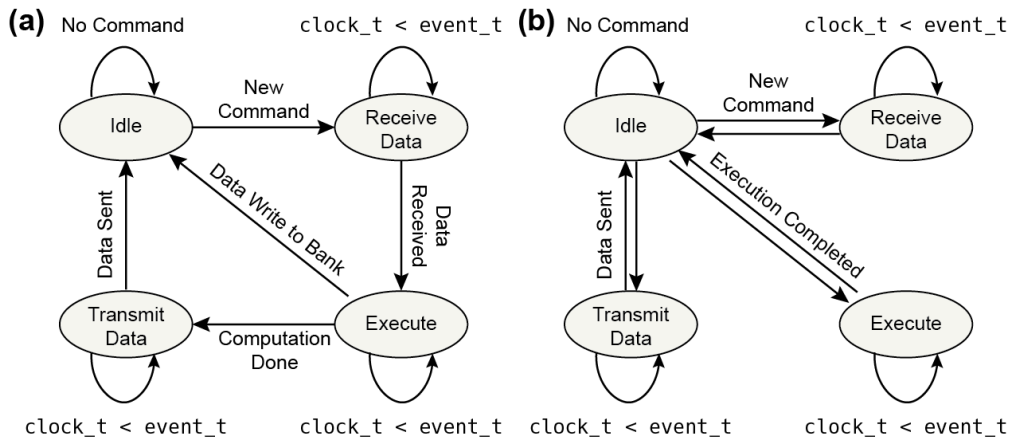


Figure 2.7: State machines of (a) DRAM-based PIM fleet, (b) host.

for data forwarding, the host needs to allocate and packet all data for a vector to DRAM, the state changes from `Execute` → `Idle` → `Transmit`.

The transition of states follows the DRAM timing constraints. At every clock cycle, the simulator checks the status of the host and the PIM fleet. If both are in `Idle` state, the current instruction is completely consumed. It then fetches the next instruction, which will be decoded into command sequences. The host chip or the PIM chip will be put into `Execute` state after the instruction is issued. The simulator will compute the firing time `fire_t` that the host or relevant PIM banks will take to complete the triggered events based on the latency model. The simulator keeps track of the status of all hardware components. If the `clock_t` reaches the `event_t`, the status of the corresponding node will be changed back to `Idle`.

The DRAM-based PIM simulator will be used in Chapter 3 to evaluate the system performance.

2.3 Event-Driven RRAM PIM Simulator – Case Studies

In this section, we brings up case studies for the RRAM-based PIM simulator on 4 different CNN models: LeNet, AlexNet, VGG-8 and VGG-11 with 2 different input sizes: 28×28 and 224×224, as summarized in Table 2.2. This analysis focuses on different ADC numbers per array and different buffer sizes, providing insights at multiple design levels for DNNs with different

structures. The default configuration of the system is adopted from Table 2.1. The RRAM PIM simulator developed here will be used in Chapter 4 to perform the power simulation.

Table 2.2: CNN benchmark architectures.

Model Name	Structure	Input Size	Kernel
LeNet	2Conv + 3FC	32×32	Large Kernel
AlexNet	5Conv + 3FC	224×224	Large Kernel
VGG-8	6Conv + 2FC	32×32	3×3 Kernel
VGG-11	8Conv + 3FC	224×224	3×3 Kernel

The CNN benchmark analysis is helpful for design space exploration of RRAM-based PIM architectures as the CNN architectures in Table 2.2 cover a wide range of CNN configurations. Deeper CNN models will have deeper channel depth as the layer increasing. This will require more crossbar arrays to map the pretrained weights. Larger input image size means there will be a lot more convolution executions, especially in the first few layers. This will potentially lead to a layer-wise execution latency mismatch. VGG models have 3×3 convolution kernel size of every layer. However, LeNet and AlexNet have larger kernel sizes in the first few layers. For example, convolution kernels of the first CNN layer are 5×5 in LeNet, and 11×11 in AlexNet. Hence, the input activation buffer needs to hold more data to initiate convolution execution. With analysis of these models, we are expected to get architecture design and resource allocation guidance for CNNs with different application features.

To start with, inference latency with different buffer sizes and ADC sharing schemes are explored. To fully consider the pipeline at inference runtime, five input images are fed to the system consecutively. The total latency is studied as shown in Figure 2.8, where N means the minimum allowed buffer size for holding data for a convolution window, as shown in Figure 2.3(a). The buffer sizes are scanned from the minimum size to the maximum. And the number of columns sharing an ADC is changed from 1 to 64, i.e. 1 ADC per column to 1 ADC per crossbar array.

From the results in Figure 2.8, we can draw three conclusions for the execution of different CNN benchmarks on RRAM-based PIM architectures. 1) According to Figure 2.8(a) and (b), increasing

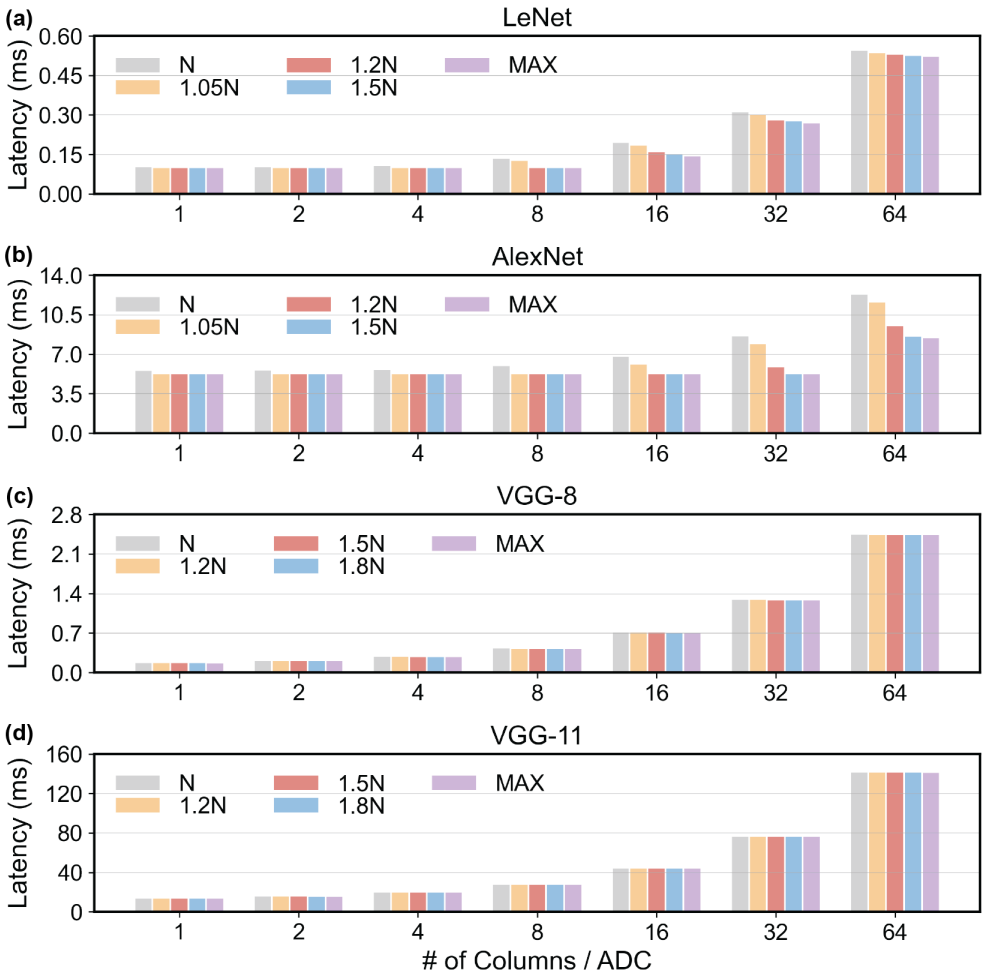


Figure 2.8: Inference latency with different buffer sizes and ADC sharing schemes of (a) LeNet, (b) AlexNet, (c) VGG-8, (d) VGG-11.

buffer size helps for CNNs with larger kernels, especially when the ADC number is limited. 2) For these CNN benchmarks, only increasing buffer sizes to slightly larger than the minimum value will lead to a decent inference latency. In both LeNet and AlexNet, the performance of 1.5N buffer size is comparable to the max buffer with capability of holding the whole input feature map. 3) According to Figure 2.8(c) and (d), increasing ADC number is more helpful for CNNs with 3×3 kernels. And these improvements can always be witnessed until each column is arranged with an ADC. However, there is a saturation effect in LeNet and AlexNet when an ADC is shared by less than 8 columns.

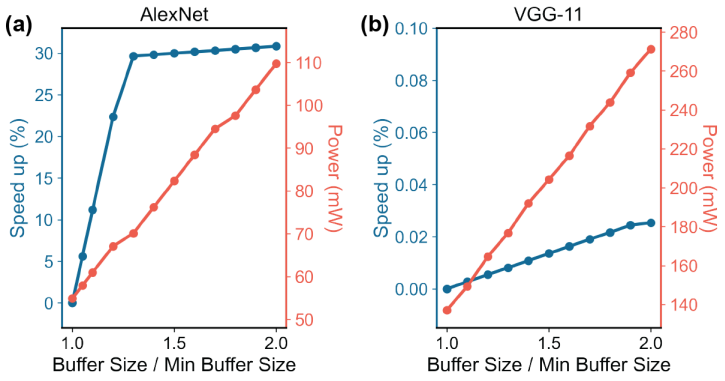


Figure 2.9: Speed up and power consumption of SRAM buffer with different buffer size for (a) AlexNet, (b) VGG-11.

Given the inspiration from above results, a fine-grained buffer size scanning is conducted to find the optimal buffer size in trade-off between inference latency and power of buffers. In this experiment, the ADC sharing scheme is fixed at 1 ADC shared by 64 columns due to the area constrain. Buffers are made up with SRAM arrays of TSMC 28 nm technology node. The power data is directly extracted from the memory cell datasheet. The speed up and power consumption of buffer with different buffer size configurations of AlexNet and VGG-11 are shown in Figure 2.9(a) and (b), respectively. For AlexNet, the speed up over baseline buffer configuration increases tremendously to 30% when the buffer size increases to 1.3× of the minimum buffer size. Further increasing buffer size only has minimum effect in performance improvement, but add great overhead in power and area of the buffer. While for the VGG-11, increasing buffer size only has negligible improvement in latency, less 0.03% of the baseline, which aligns with the results in Figure 2.8(d). However, for deeper CNN models, later convolution layers will require large buffers to hold input feature map with large channel depth. This will increase power and area greatly.

2.4 Summary

PIM systems have the potential to accelerate data intensive tasks including machine learning models, and previous simulation and real-chip measurements have proved their effectiveness in

VMM acceleration. However, careful system-level considerations need to be performed to verify their practical impacts. In this chapter, we introduce the full system, clock-cycle-accurate PIM system simulators driven by the data flow event. The initial simulation results give fruitful design considerations at the architecture, circuit and model levels. At the system level, many factors will constrain the data communication and execution, and simulation strategies proposed in this chapter helps both the design and evaluation of PIM systems. These simulators will be as the foundation of Chapter 3 to 5 in this thesis.

CHAPTER 3

PIM-GPT: A Hybrid Process-in-Memory Accelerator for Autoregressive Transformers

3.1 Background and Motivation

3.1.1 Background

Attention-based Transformer models have revolutionized natural language processing (NLP) by capturing long-term dependencies in the input data [96]. Transformer models including GPT and BERT have demonstrated superior performance in many NLP tasks such as text generation [97][98], text classification [99][100][101], and machine translation [102][103] compared to convolution neural networks (CNNs) or recurrent neural networks (RNNs) [97][104][105]. GPT in particular has attracted widespread public interest in text generation. It is a decoder-only Transformer model that generates context in an autoregressive manner by producing a single token at one time [98]. However, the sequential processing feature of GPT will result in notable under-utilization of parallel processing resource on the GPU, particularly when accelerating small batch inference.

Compared to CNNs, GPT has two main features: 1) extremely large model size and 2) low compute-to-memory-ratio. As shown in Figure 3.1, the GPT3-XL model consists of 1.15 billion parameters [105], more than a hundred times higher than common CNNs such as ResNet-18 [106], while the arithmetic intensity per parameter (ops/parameter) is only 2.1, much lower than 48.4 in ResNet-18. These features impede the efficiency of GPU and other accelerators. The on-chip

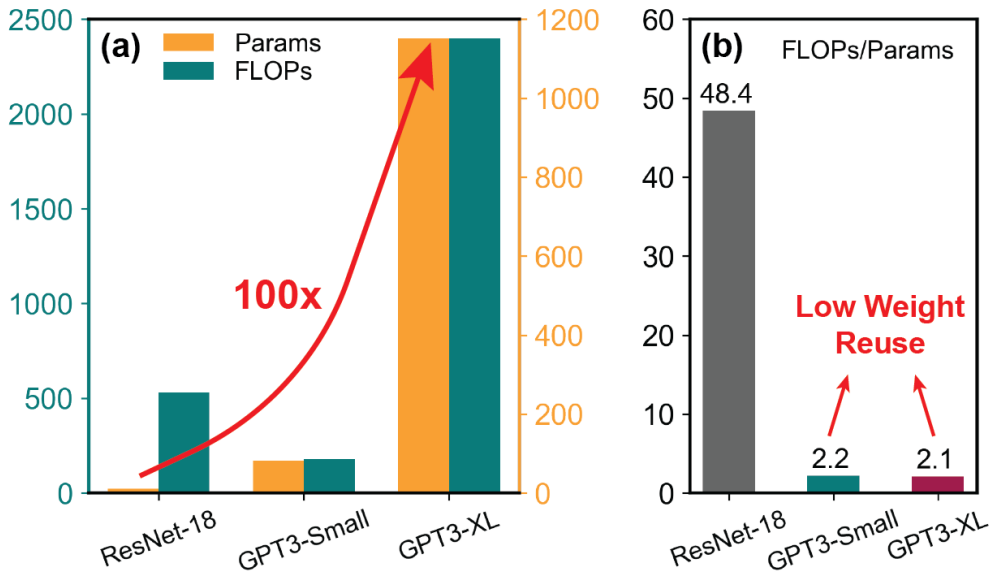


Figure 3.1: (a) Parameter and computation cost comparison of GPTs and ResNet-18. (b) Operation/parameter ratios for CNN and GPT models.

memory is insufficient to store all weights in large GPT models, leading to extensive off-chip memory access. This imposes penalties in both performance and energy consumption. In the context of single token generation in GPT, self-attention relies on vector-matrix multiplication (VMM) instead of matrix-matrix multiplication used for processing the entire sentences. Self-attention, along with feed-forward network (FFN) are based on VMM, characterized by low data reuse as the weight value in the matrix are employed only once, in contrast to reuse the same weight multiple times in convolution operations.

Recently, several Transformer accelerators have been proposed to accelerate GPT inference [20][107][108]. However, these designs generally suffer from the following drawbacks: 1) expensive hardware overhead such as the usage of HBM and dense in-memory logic; 2) model customization such as model and token pruning, which will make the architecture less flexible and cause accuracy loss; 3) incapability of long token length generation due to inefficient intermediate data management; 4) lack of end-to-end acceleration as most studies only focus on attention computation and feed-forward layers. Moreover, many of the existing Transformer accelerators are only designed for encoder-only model like BERT, instead of decoder-only models like GPT

[109][110][111][112][113][114][115].

DRAM-based PIM architecture is a promising architecture to accelerate memory-bounded tasks [3][116]. The high storage capacity of DRAM allows all model parameters to be stored. Integrating computing elements onto the DRAM chip, PIM enables the localized consumption of data, utilizing high internal bandwidth and minimizing the need for external DRAM data access. The placement of computation units for MAC operation can be distributed across various regions, including sub-array, bank, I/O driver and logic die. Among these choices, siting MAC units at the bank level provides a favorable balance between performance, energy consumption and area cost [117]. Recent PIM developments further demonstrate the efficient acceleration of MAC operation with integrating computation components at bank level of DRAM chips [2][13][14][15]. However, only limited, low-density logic circuits can be fabricated on DRAM chips due to the significant constraints in memory fabrication process [6][118]. Developments to date are limited to generic MAC operations, and efficient end-to-end PIM acceleration of GPT inference still needs to be developed with practical and feasibility considerations.

In this chapter, we will discuss PIM-GPT [32], a complete solution for GPT inference acceleration. At the hardware level, PIM-GPT is a hybrid system that includes DRAM-based PIM chips to accelerate VMM near data and an application-specific integrated circuit (ASIC) to support other functions that are too expensive for PIM including necessary non-linear functions, data communication and initiating instructions to the PIM chips. At the software level, mapping scheme is optimized to efficiently support the GPT dataflow. To accommodate the large model size and improve performance, the computation workloads are evenly distributed across PIM channels and banks to maximize the utilization of available computation resources and on-chip bandwidth. A high-level mapping scheme for VMM operation is shown in Figure 3.2. As each row of the matrix must be multiplied by the same vector, the rows are distributed across all banks, as indicated by colors. When the VMM begins, the vector is broadcasted to all banks and multiplied with matrix data from each bank in parallel. If the matrix dimension exceeds the physical storage of bank row, it will be divided into chunks for mapping and computation. Multi-attention heads are concatenated

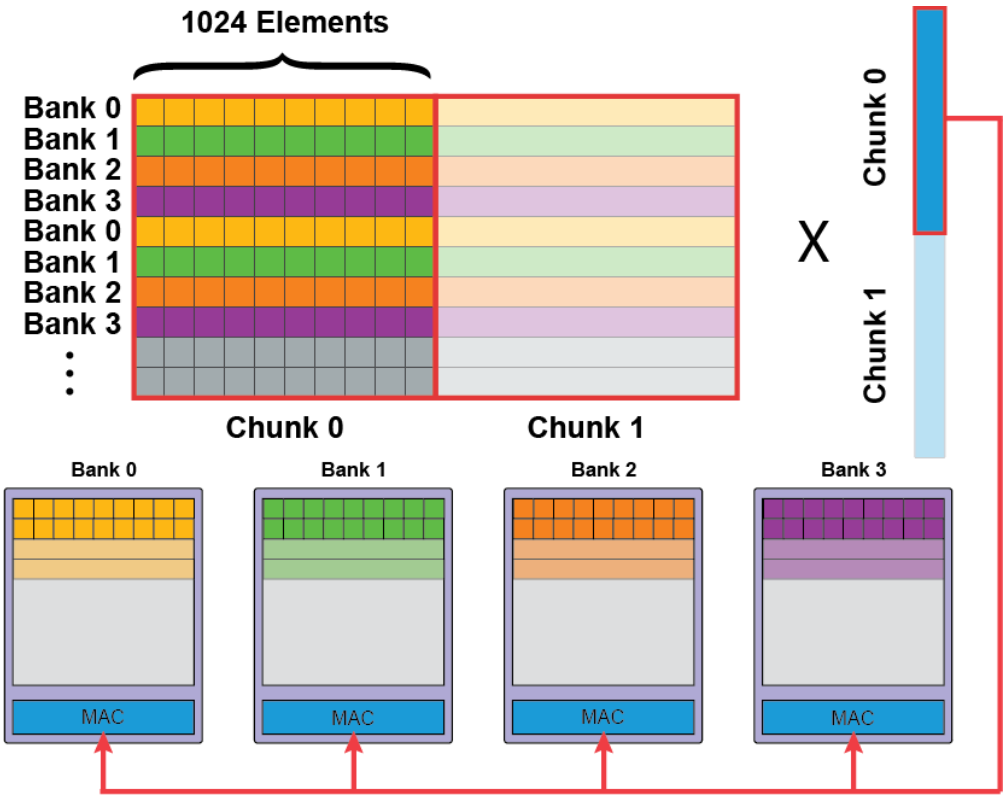


Figure 3.2: Mapping scheme for VMM operation.

to large matrix to utilize parallel computing capability as well as maximize data locality. Compared to existing Transformer accelerators [20][107][108], the proposed PIM-GPT supports large GPT models end-to-end without the need of expensive HBM, making it an efficient and practical solution for GPT acceleration. Benchmarking analysis shows the proposed PIM-GPT achieves state-of-the-art speedup and energy efficiency for GPT inference tasks.

3.1.2 Transformer Models

Figure 3.3 illustrates the typical structure of a Transformer model. Different from CNNs and RNNs, the Transformer model uses a self-attention mechanism that captures the relationship between different words in the whole sentence [96]. The original Transformer model consists of an encoder and a decoder, both containing N number of identical transformer blocks. Each block includes a self-attention module and a feed-forward network (FFN). Among them, BERT and GPT

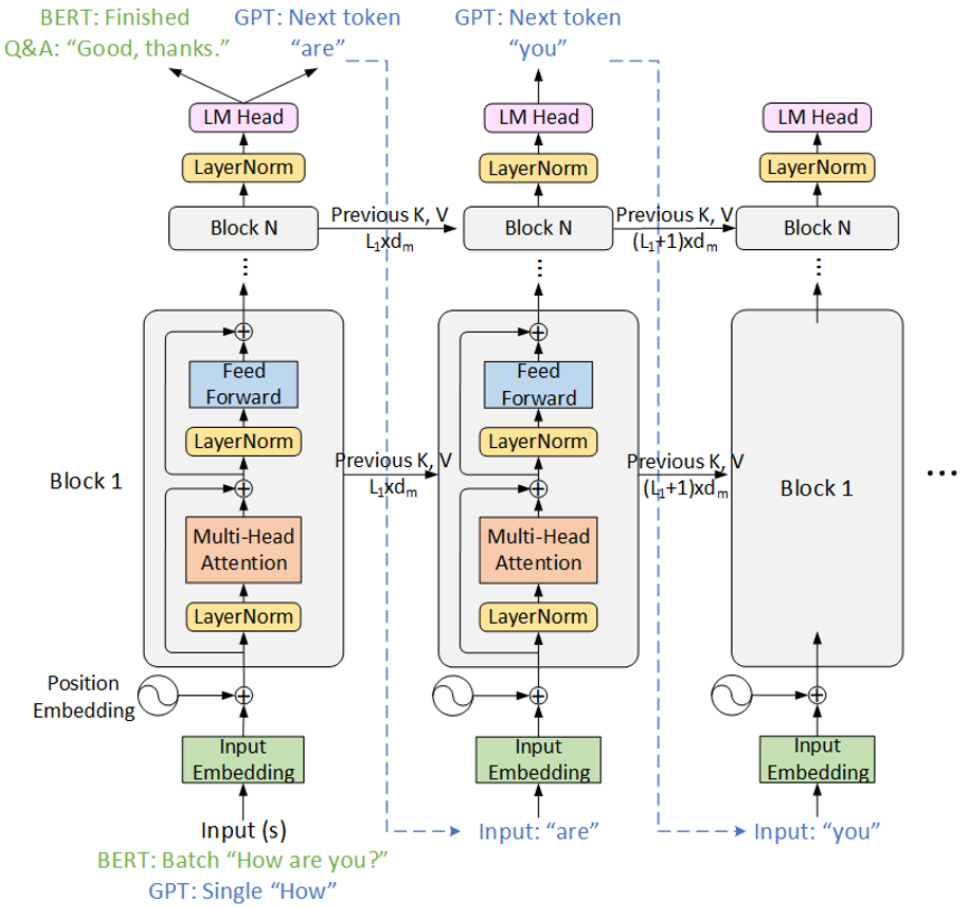


Figure 3.3: Transformer model architectures of BERT and GPT.

are the two most popular language models. BERT is an encoder-only model, which processes all input tokens at once and attends to content in both directions [97]. In contrast, GPT is a decoder-only model, which typically handles a single token at one time and generates the next token in a sequential manner by attending to all previous tokens [105]. Despite the differences, these two models share similar blocks, as shown in Figure 3.3. For GPT, the input token is first transformed into a vector of dimension d_m by the input embedding layer, where d_m is the feature dimension of the model. Positional information of the token with respect to the sentence is also added to the input embeddings. The processed token is then fed into the Transformer blocks.

The input token is first multiplied with three linear transformation matrices ($W_{K,Q,V}$) to obtain Query (q) , Key (k) and Value (v) vectors, where $W_K \in \mathbb{R}^{d_m \times d_k}$, $W_Q \in \mathbb{R}^{d_m \times d_k}$, $W_V \in \mathbb{R}^{d_m \times d_v}$.

The current Key and Value vectors k, v are then concatenated to the key, value matrices computed from the previous inputs to form the updated Key, Value matrices. The q vector and Key, Value matrices are then passed to the self-attention head to capture the dependencies between tokens with the following equation:

$$\text{Attention}\,(q, K, V) = \text{softmax}(\frac{qK^T}{\sqrt{d_k}})V \tag{3.1}$$

The scaled dot product between q and K^T is first computed to obtain the attention score, which measures the relation between the current token and all previous tokens. Then a softmax operation is applied to normalize the attention score between 0 and 1. In the next step, the attention score is multiplied with the value matrix to produce the head output. To allow the model to learn different relationships for each token, multi-head attention technique is adopted. The input vectors are split across attention heads, and each chunk goes through a separate head in parallel. All head outputs are combined by a linear projection layer to produce the final attention output.

Following the multi-head attention, the attention output is fed into a FFN network, which consists of two fully-connected networks with Gaussian Error Linear Unit (GELU) activation function [119] in between. The attention and FFN layers both contain a layer normalization and a residual connection, as shown in Figure 3.3. The output from the attention block is then applied as inputs to the next attention block for subsequent processing, and the process is repeated through N attention blocks. Afterwards, a final output layer is used to predict the next token. The content is generated autoregressively by repeating this process until it reaches the required token length.

Unlike GPT, BERT processes all input tokens in parallel and produces the outputs at once. Therefore, the core computation is matrix-matrix multiplication, and the performance is computation-bounded. In contrast, for GPT the core computation is VMM and the arithmetic intensity is relatively low but the required memory access is high. As a result, throughput optimized architectures such as GPUs are not efficient for GPT inference, while PIM techniques that leverage compute capabilities on DRAM chips are promising for GPT inference hardware acceleration.

3.1.3 Motivation

As shown in Figure 3.1, the computation primitive of GPT has a low ops/parameter ratio and is memory-bounded since it processes and generates a single token at one time. Compared to computation-bounded models, the sequential feature of decoder cannot utilize parallel processing cores in GPU efficiently, particularly for inference tasks without batching. Recently, several Transformer accelerators have been proposed. However, most of them still lack end-to-end model acceleration of GPT autoregressive token generation. And these works still suffer from intensive data transfer of large weight matrices. The on-die data movement and DRAM interface have a significant energy consumption overhead [1][120].

PIM is a promising approach to relieve the memory bottleneck by storing matrix and performing computation in memory. With weight matrices stationary in memory, only the input and output vectors will be transferred though the DRAM interface. Hence, the memory access complexity can be reduced from $O(n^2)$ to $O(n)$. Instead of incessantly adding computation units to DRAM to support all instructions, it is more efficient to execute non-VMM operations in a separate custom designed chip for achieving end-to-end GPT acceleration. At the software level, efficient workload distribution and dataflow management are required to fully exploit the advantage of PIM. The proposed PIM-GPT is such an end-to-end GPT accelerator with practical considerations in hardware implementation.

3.2 PIM-GPT Architecture

PIM-GPT is a memory-centric acceleration system aimed to support Transformer-based autoregressive token generation models including GPT. The PIM-GPT system is shown in Figure 3.4, which composes of PIM chips and a custom designed ASIC. The design principle of PIM is to maximally leverage data locality and parallelism to achieve high system performance and energy efficiency during VMM in attention and FFN. To achieve efficient VMM, PIM-GPT strategically partitions GPT model, taking hardware resources into account. It achieves parallel computing

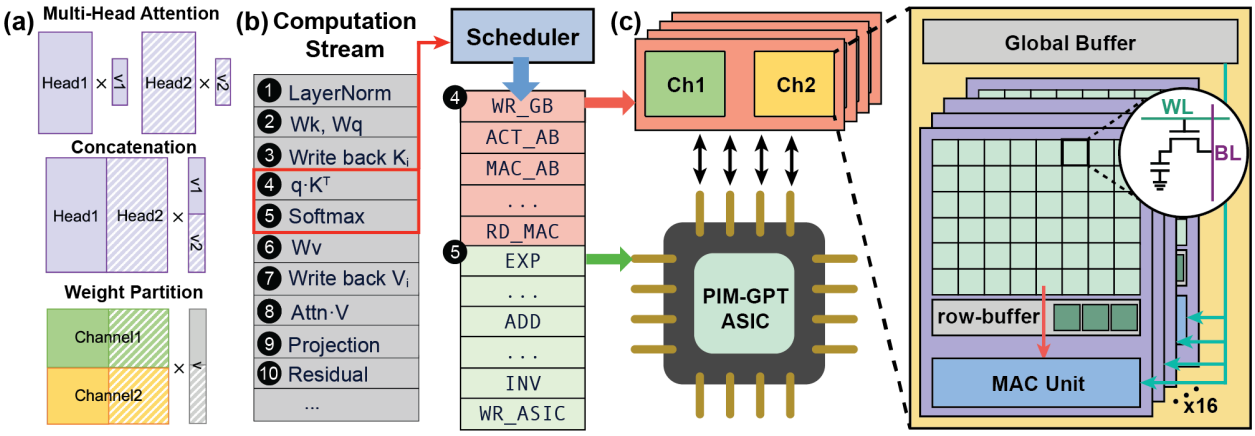


Figure 3.4: PIM-GPT system overview. (a) Hardware-aware GPT model partition. (b) Compilation of computation stream to command stream. (c) PIM-GPT hardware architecture.

by broadcasting a single vector and reducing instruction overhead. PIM-GPT employs a specific mapping scheme for attention mechanism, as shown in Figure 3.4(a). Weight values from multiple attention heads are concatenated to accommodate the physical capacity of DRAM banks. The concatenated attention matrices, along with weights in FFN layers, are distributed to all channels and banks for parallel operation, following the mapping scheme in Figure 3.2, details of which will be elaborated in Section 3.3. PIM-GPT integrates 8 channels. And all chips are equipped with MAC units, executing VMM locally by broadcasting the vector from the global buffer and loading matrices from DRAM bank arrays, as shown in Figure 3.4(c).

To facilitate end-to-end acceleration of large GPT models, non-linear functions are executed on the ASIC chip. It is essential to highlight that PIM-GPT targets on eliminating memory access of matrix data, requiring only the transfer of input/output vectors between PIM and ASIC for downstream computations, as well as data communication and intermediate data storage. This integrated approach leverages the strengths of both PIM and ASIC, optimizing their capabilities to accelerate various computation tasks in the GPT computation stream with minimized data movement between them. Figure 3.4(b) illustrates the compilation of ❹Attention and ❺Softmax to command streams for PIM and ASIC, respectively. All data in PIM-GPT are in bfloat16 (BF16) format, which preserves the approximate dynamic range of 32-bit floating point number to balance performance and accuracy.

Table 3.1: PIM-GPT baseline hardware configuration.

Timing Constraint	tRCD=12ns, tRP=12ns, tCCD=1ns, tWR=12ns, tRFC=455ns, tREFI=6825ns
IDD	IDD2N=276mA, IDD3N=262mA, IDD0=366mA, IDD4R=1590mA, IDD4W=1410mA, IDD5B=831mA
GDDR6 Specification	Channel = 8, Banks/channel = 16, Capacity/channel = 4Gb, Row size = 2KB, Column number = 16k, Frequency = 1GHz, Pins = 16/channel, Data rate = 16Gb/s/pin, I/O Power = 5.5 pJ/bit
PIM	Buffer = 2KB/channel, MAC unit = 1/bank, Frequency = 1GHz, Power = 149.29mW
AISC	Technology 28nm, Frequency = 1GHz, SRAM = 128KB, # of Adders = 256, # of Multipliers = 128, Area = 0.64mm^2, Power = 304.59mW

3.2.1 DRAM-based PIM for VMM

Placing MAC units and SRAM buffer inside DRAM modules has been widely used in PIM architectures for machine learning applications to minimize the memory access [13][14][117][121]. MAC units can be placed at sub-array level or bank level. But for the practical architecture design, we need to balance the budget for internal bandwidth utilization and area overhead. Residing MAC units at sub-array has the bandwidth advantage since it is the closest place to data. However, this will add intolerable area overhead, up to 120% [117]. Moreover, sub-array level modification requires complex DRAM command redevelopment. Hence, PIM-GPT placing the MAC units at every bank. And all 16 banks can perform MAC simultaneously, as shown in Figure 3.5(a). Such MAC units arrangement has been proved feasible in both GDDR6 and HBM prototypes [13][14]. Take GDDR6-based PIM as an example, placing MAC unit per bank offers 512 GB/s peak bandwidth per channel at 1 GHz (256 bits/bank $\times$ 16 banks). SRAM buffer has been used to store and broadcast the input vector. However, integrating large SRAM buffer into DRAM is expensive and impractical. To provide a practical solution of GPT acceleration, we adopt 2 KB buffer size from [13], which will broadcast vectors to all MAC units simultaneously. When the input length exceeds the buffer size, vectors will truncated into chunks, as shown in Figure 3.2. Partial MAC results will be computed and forwarded to the SRAM on ASIC, followed by downstream partial sum execution on the ASIC.

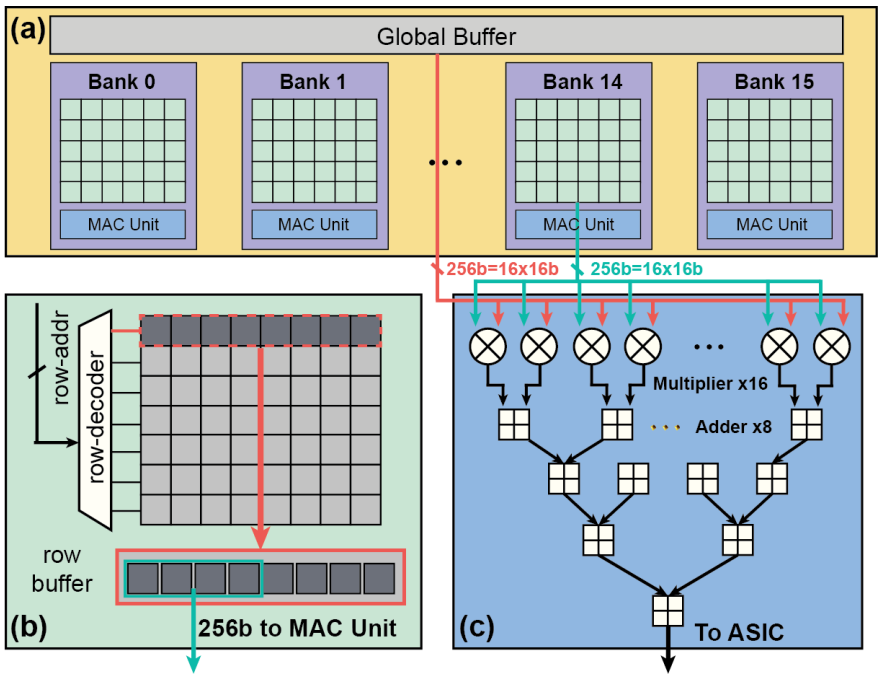


Figure 3.5: DRAM PIM organization. (a) A channel is composed of a global buffer and 16 banks. A bank contains (b) a conventional DRAM bank and (c) a MAC unit with multipliers and an adder tree.

Compared to writing back to DRAM, forwarding the subportion of VMM results to ASIC reduces overall latency. Eliminating writing back also frees DRAM banks to perform subsequent parallel VMM.

The bank organization is identical to conventional DRAM architectures, as shown in Figure 3.5(b). Once a row address is decoded, the entire corresponding row will be activated and all stored data will be forwarded to the row buffer. Herein, if the bank is not closed (precharged), data will be preserved in the row buffer. Reading data from the row buffer is much faster than from the bank array, since it skips the long latency row activation step. Hence, to maximize the utilization of peak internal bandwidth, data should be consumed from the row buffer as extensively as possible. This requires bank scheduling adheres to the open-row policy, wherein a row is not immediately precharged after data access. By employing this policy and appropriately mapping weight data, the MAC unit can more rapidly consume data from the bank. In our optimized mapping scheme, matrix rows for VMM are partitioned and directly mapped to the same row addresses, thereby maximizing

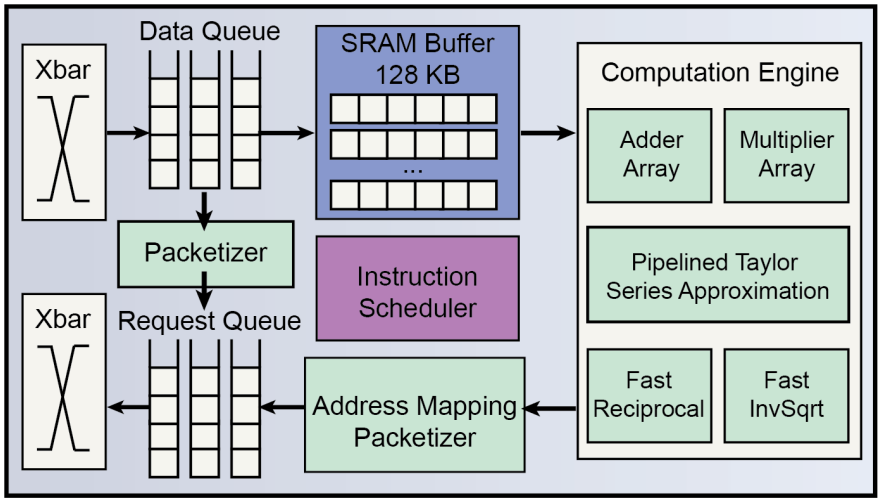


Figure 3.6: ASIC architecture of the PIM-GPT system.

the row hit rate, as outlined at a high level in Figure 3.2. Further details will be discussed in Section 3.3.

Inside each bank, only multipliers and an adder tree are implemented for MAC operation, as shown in Figure 3.5(c). Similar architecture has been proven to be effective in prior designs [13][121][122]. During MAC operation, 16 vector values and corresponding weights are fetched from the buffer and banks in the PIM channel, respectively. The 16 multipliers multiply the vector data with weights. The adder tree accumulates the multiplication results for downstream computation, as shown in Figure 3.5(c). The MAC units are operated in a pipelined fashion to maximize the throughput of the system, i.e. once the multiplication is done, the multipliers fetch the next chunk of vector and weight in the next clock cycle. The same principle is used for the adder stages. To minimize hardware cost and improve efficiency, PIM-GPT only performs VMM operations in the PIM while assigning all other computations, such as data bypassing, division and activation functions, to the ASIC. By doing so, the design allows the integration of lightweight MAC units to the DRAM chip to consume data locally without significantly sacrificing the memory capacity.

3.2.2 ASIC Architecture

The ASIC in PIM-GPT is used to manage data communication and support non-VMM arithmetic computations and intermediate data storage. The ASIC architecture is shown in Figure 3.6. Since VMM operations are performed in the PIM channels, data will only be read from DRAM PIM when a VMM operation is done and requires downstream computation or communication. The PIM channels communicate with the ASIC through the memory bus and crossbar interconnects. The interconnects support data fetching from any DRAM channel and sending memory requests to a single channel or broadcasting data to all channels after packeting data with address. Data read from PIM have two possible paths on the ASIC: 1) writing back to banks in other PIM channels, such as Key, Value matrices for subsequent VMM operations, and 2) going through computation blocks in the ASIC, such as layer normalization, softmax, etc.

If the data require downstream computation, they will be temporarily stored in the on-chip SRAM buffer. The SRAM buffers and computation engines in the ASIC of the proposed PIM-GPT are designed for billion-parameter level models such as GPT3-XL. For smaller models or instructions that only utilize portions of the computation resources, power gating schemes will be applied to SRAM arrays or unused computation blocks to lower the ASIC power consumption.

The adders and multipliers in the computation engines follow the standard floating-point unit design to support summation and multiplication. For design reuse and performance considerations, other computation tasks are all implemented with approximation algorithms using only addition and multiplication to achieve the required precision.

Three functions that require approximation are shown in Equation 3.2−3.4: softmax, layer normalization and activation function GELU further approximated using Equation 3.4.

$$s(x_i) = \frac{e^{x_i}}{\Sigma_{j=1}^{N} e^{j}} \tag{3.2}$$

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \times \gamma + \beta \tag{3.3}$$

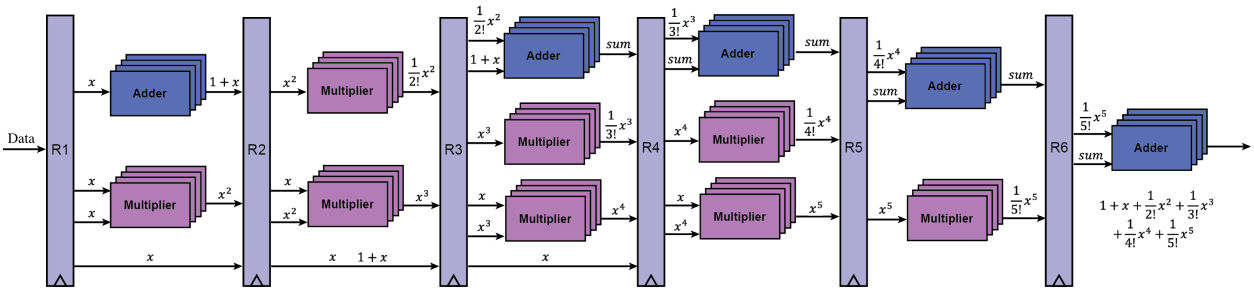


Figure 3.7: Pipelined Taylor series approximation scheme.

$$\begin{aligned} \mathrm{GELU}(x) &= \frac{x}{2} \times [1 + erf(\frac{x}{\sqrt{2}})] \\ &= \frac{x}{2} \times [1 + \tanh \sqrt{2/\pi}(x + 0.044715 \times x^3)] \end{aligned} \tag{3.4}$$

The nonlinear function e^x, $\tanh x$, division and square root in these functions cannot be naively computed using addition and multiplication. Under given precision and data range, they can be efficiently approximated and converge in rapid iterations. Here e^x and $\tanh x$ are computed using Taylor series approximation with the first six items, which can be computed with addition and multiplication, as shown in Equation 3.5 and 3.6. To improve the throughput, a pipelined Taylor series approximation block is implemented in the ASIC. Figure 3.7 shows how the e^x is computed by the Taylor series approximation.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + O(x^6) \tag{3.5}$$

$$tanh(x) = x - \frac{x^3}{3} + \frac{2x^5}{15} + O(x^7) \tag{3.6}$$

The division operation is computed by multiplying the numerator with the inverse of the denominator. Both the reciprocal and inverse square root operations can be calculated with addition and multiplication following Newton's method. A proper initial value is required to ensure the fast convergence. For reciprocal, we take advantage of Newton-Raphson division algorithm as shown

in Figure 3.8. The fast inverse square root algorithm is adopted from Quake III Arena's source code [123], as shown in Figure 3.9.

Algorithm 1: Newton-Raphson Division

Data: `BFloat16` $D = (S)M \times 2^E$
Result: `BFloat16` $\frac{1}{D}$
/* Scale by exponent subtraction */
1 $D' = D/2^{(E+1)}$
2 $X = \frac{48}{17} - \frac{32}{17} \times D'$
3 **for** *iter* in $\lceil \log_2 \frac{P+1}{\log_2 17} \rceil$ **do**
4 $\quad X = X + X \times (1 - D' \times X)$
5 **end**
/* Scale the result */
6 $X' = X/2^{(E+1)}$

Figure 3.8: Newton-Raphson division algorithm.

Algorithm 2: Fast Inverse Square Root

Data: `BFloat16` $D = (S)M \times 2^E$
Result: `BFloat16` $\frac{1}{\sqrt{D}}$
1 $D' = D \times 0.5$
/* Unpack data and pad with 0 */
2 `uint32_t` $L \leftarrow \{\text{unpack}(D'), \text{0x0000}\}$
3 $L' = \text{0x5f3759df} - L >> 1$
/* Keep 16 high bits */
4 `BFloat16` $X \leftarrow \text{pack}(L')[31:16]$
5 **for** *iter* in IterNum **do**
6 $\quad X = X \times (1.5 - D' \times X \times X)$
7 **end**

Figure 3.9: Fast inverse square root algorithm.

The Newton-Raphson Division algorithm is friendly for floating point numbers since it requires the input D to scale to a value close to 0, which can be easily done with exponent subtraction and mantissa shift in floating point data format. The P in line 3 is the precision of P binary places. Hence, for a 16-bit floating point number, it will take three iterations to get an accurate result.

The fast inverse square root algorithm unpacks the BF16 data into 16-bit integer data and padding 16-bit zeros to get an accurate approximation, followed by shift and subtraction from a constant. The pack step utilizes the 16 high bits of INT32 L' to assemble a BF16 data's sign bit, exponent and mantissa. In the fast square root algorithm, it can converge in a single step iteration. Here we take a conservative two step iteration.

3.3 PIM-GPT Dataflow

PIM-GPT distributes workloads among all PIM banks and ASIC efficiently. For VMM operation, the input vector is stored in the ASIC SRAM buffer and broadcast to the buffer of all PIM channels, as depicted in Figure 3.10. All MAC units will execute the same MAC instruction on different matrix partitions in parallel to fully utilize the PIM computation resources without instruction overhead. The subvectors from channels will be sent back to ASIC for concatenation and downstream communication and computation. PIM-GPT implements the following techniques to coordinate workload between the PIM channels and the ASIC: 1) The partial outputs of VMM can be forwarded to the ASIC before the whole computation is completed, which effectively eliminates the data write back to DRAM banks; 2) When input vector length exceeds the buffer size, SRAM buffer on the ASIC are reserved to store intermediate data and the ASIC will accumulate partial VMM results from DRAM; 3) Pipelining between data transmission and computation, i.e. the ASIC will start operations on partially received vector while the rest are in transmission.

The model mapping includes storing the weights to the allocated banks, as well as reserving space for the intermediate data (Key, Value matrices) for attention computation since they are dynamically expanded with token generation. To enhance the system performance, the mapping scheme is optimized to: 1) maximize row hit rate by exploiting data locality; 2) increase computational parallelism by balancing the workload across DRAM banks; and 3) reduce latency by minimizing data movement. During runtime, the system automatically computes the bank address in the reserved space to write back the generated Key and Value vectors. The high-level description of

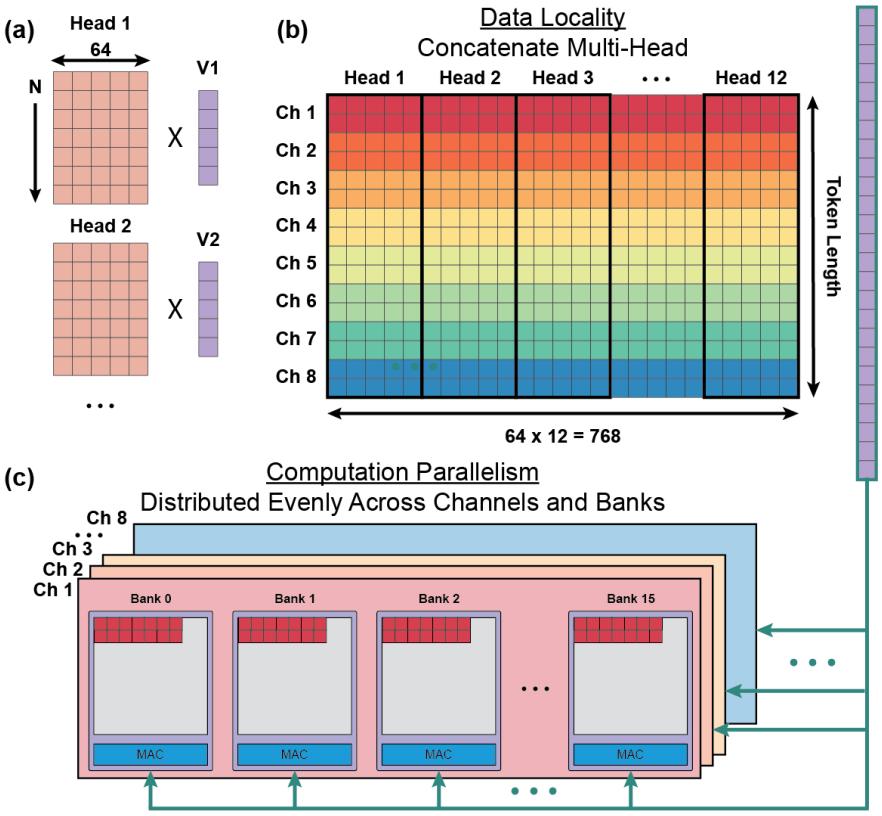


Figure 3.10: Mapping strategy for Key Matrix. (a) Multi-head attention. (b) Concatenate multi-head to exploit data locality. (c) Distribute weight matrix evenly across channels and banks to maximize computation parallelism.

the mapping scheme is shown in Figure 3.11.

3.3.1 Attention Head Mapping

As shown in Figure 3.10, the mapping scheme leverages data locality and maximizes computation parallelism. Since activation (ACT) and precharge (PRE) commands are expensive in both latency and energy, achieving a high row-hit rate is preferred. To this end, matrix data used for MAC operations need to be mapped to consecutive physical DRAM cells. This approach means the corresponding row only needs to be activated once to transfer all required data to the row buffer, and the MAC units can keep consuming data already in the opened row to minimize ACT and PRE operations.

To take advantage of the data locality, it is desired that a row is fully mapped with data. However,

Algorithm 3: Model Mapping to PIM Banks

Data: Computation Graph; PIM Configuration
Result: Memory Mapping and Reservation

```
/* Map weights to PIM banks          */
for vmmBlock in Computation Graph do
    if vmmBlock.multiHead then
        hitScore ← maxRowHit(nhead, ncol)
        vmmBlock ← concat(hitScore, vmmBlock)
    end
    Mapping ← maxParallel(vmmBlock, nch, nbank)
end
/* Reserve PIM bank rows for kv      */
for wrBlock in Computation Graph do
    if block == write_k then
        hitScore ← maxRowHit(nhead, ncol, ltoken)
        wrBlock ← concat(hitScore, wrBlock)
    end
    Mapping ← Reserve(wrBlock, ltoken, nch, nbank)
end
```

Figure 3.11: PIM-GPT mapping algorithm.

a single attention head can be much smaller than the DRAM array dimension. As shown in Figure 3.10(a), attention head width of GPT2-small is 64 while a bank row can store 1024 16-bit data. To maximize the row hit rate, all attention heads in the same layer are concatenated to fill up the DRAM bank. Take GPT2-small as an example, 12 attention heads are concatenated to a wider matrix with the width of 768, along with the concatenation of the input vector, as shown in Figure 3.10(b). To maximize the utilization of MAC units, rows of the matrix are evenly distribute across PIM channels and banks, as indicated by the rainbow color in Figure 3.10(b) and (c). Figure 3.10(c) is a detailed example showing how K matrix are mapped through 8 PIM channels, assuming the token length is 256 in GPT2-small model. First, attention heads in a layer are concatenated along column direction to form a larger matrix, with the dimension of 256×768. The concatenated matrix is mapped following the row major approach and evenly distributed 32 matrix row to all available channels, as indicated by the rainbow colors in Figure 3.10(b) and (c). Inside each channel, all 16

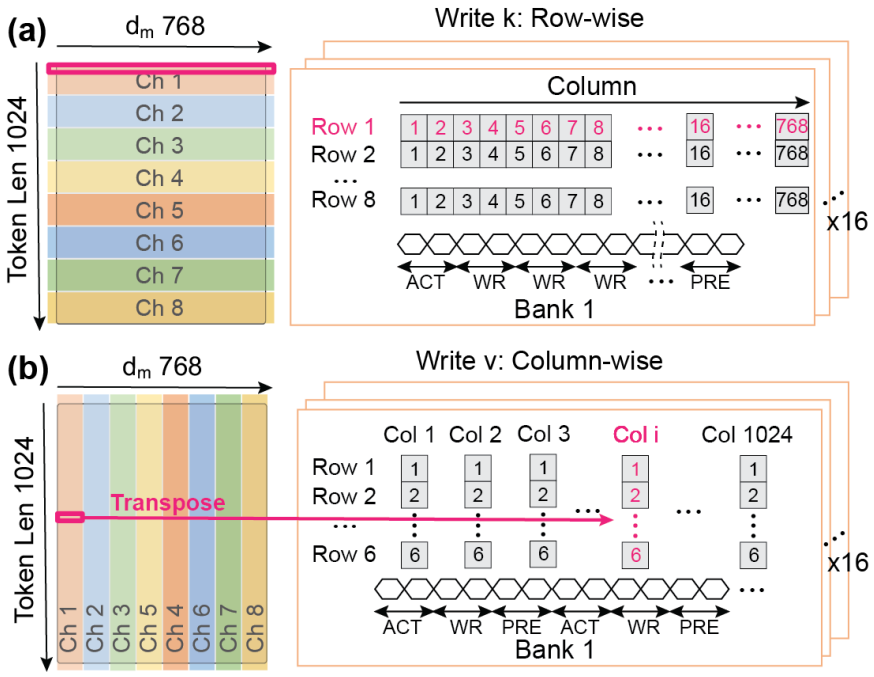


Figure 3.12: (a) Write k row-wise. (b) Write v column-wise.

banks are mapped with 2 rows of the matrix and execute MAC operations with the same vector in parallel. The computation is similar when it comes to VMM in the attention projection and FFN layers. This mapping scheme distributes matrices evenly to achieve the highest possible DRAM channel-wise and bank-wise parallelism and run as many MAC units at the same time as possible.

3.3.2 Intermediate Data Memory Reservation

Key and Value results need to be written back to the PIM banks and append to the existing Key and Value matrices. In the mapping stage, PIM-GPT reserves the required space in PIM banks for these intermediate data. The write back scheme of key and value results are shown in Figure 3.12(a) and (b), respectively. Key and Value write back are in row-major and column-major, respectively, since the transpose of Key matrix is required in Equation 3.1, while no transpose of Value matrix is required.

PIM-GPT exploits data locality during write. During runtime, the Key vectors produced by multi-attention heads will be concatenated together, which is corresponding to $N = 1$ in Figure

3.10(a). As shown in Figure 3.12(a), Key vectors with length of 64 from 12 heads are concatenated to form a vector with length of 768, and written to the corresponding PIM bank row reserved for the current token, as highlighted by magenta in Figure 3.12(a). The write command can be executed consecutively after one ACT to store the whole Key vector, as shown in the timing diagram in Figure 3.12(a). The concatenated Key vectors produced by all token generation steps are evenly stored across all channels, as indicated by the color blocks in Figure 3.12(a). Within each channel, they are evenly distributed to all banks.

Value results are stored in column-major fashion. As shown in the timing diagram of Figure 3.12(b), in this case we can only write one data in an activated row. Then we close the row and move to the next row for the next data. Hence, data locality cannot be leveraged in Value result write back. To maximize the write throughput and computation parallelism in the subsequent VMM step, we distribute the Value matrix to all channels and banks, as shown in Figure 3.12(b).

3.3.3 Weight Matrix Tiling

For larger GPT models, widths of both concatenated multi-head matrices and weight matrices in FFN layers can exceed the capacity of a bank row. In this circumstance, the input vector length also exceeds the buffer size. Hence, both the matrix and the vector need to be truncated for VMM operation as shown in Figure 3.2, followed by accumulating the partial result. The partial sum is executed in ASIC for alleviating DRAM to execute VMM restlessly in the meanwhile. Take the input length of 2048 in Figure 3.2 as an example, the matrix and the vector are sliced into two chunks. The elements in the same row will always mapped to the same bank, as indicated by the faded color. Computation on the second chunk starts after the first chunk is complete to avoid frequently overwriting the SRAM buffer.

3.4 System Evaluation

3.4.1 Evaluation Method

As DRAM fabrication process is highly constrained compared to the CMOS process, we refer GDDR6-based PIM prototype reported from SK Hynix for performance evaluation, since it only integrated lightweight MAC units near banks. The area of one processing unit (PU) is 0.19 mm^2 [13]. All logic components of the ASIC are synthesized with SystemVerilog using Synopsys Design Compiler at TSMC 28nm HPC+ process node. Area and power of the logic components are obtained from the synthesis results. The area and power of SRAM buffer are extracted from the TSMC 28nm datasheet based on Synopsys Memory Compiler. The ASIC only consumes a core area of 0.64 mm^2, and the peak power is 304.59 mW.

For DRAM energy benchmark analysis, we synthesized the optimized MAC units at 28 nm technology, followed by scaling the voltage to 1.25V to match the GDDR6 supply voltage [13]. Since routing is more complex in DRAM due to the limited metal layers compared to CMOS logic process, we conservatively multiply the power by 1.5, which comes to 149.29 mW for 16 MAC units. Since GDDR6 can be fabricated in 1ynm technology [13], the actual power consumption is expected to be lower than this estimate. Table 3.1 lists the timing constraints and current values used to model the PIM behavior for each command. For PIM related commands, the timing constraints are obtained from [13]. For normal DRAM commands, we adopt GDDR5 timing constrains in [95] to make a conservative estimation due to the lack of detailed information of GDDR6. Similarly, the current values are obtained from DDR5 datasheet [124] and multiplied by 3 due to the current consumption increases during all bank parrallel operation [15]. The GDDR6 I/O access energy 5.5 pJ/bit is adopted from GDDR6 datasheet [125]. The system performance in latency and power efficiency is evaluated based on these conservative assumptions. Detailed hardware configuration is summarized in Table 3.1.

The performance and energy efficiency of the proposed PIM-GPT system are evaluated using the simulator and compared to GPU (NVIDIA T4) and CPU (Intel Xeon Gold 6154 with 16Gb

DDR4). 4 GPT2 [104] and 4 GPT3 [105] models with up to 1.4 billion parameters are implemented on the proposed PIM-GPT system, and used for the benchmark analysis, as summarized in Table 3.2. The simulator reports the total latency and the command sequences that are performed by the PIM channels and the ASIC. For PIM power, we multiply the IDD values consumed during each command with the corresponding latency and VDD, following the standard procedure [124][126]. DRAM refresh operations are also included. The energy consumed by the PIM MAC units and by the ASIC are computed by multiplying the latency reported by the simulator with the synthesized power consumption.

Table 3.2: Sizes, architectures, and floating points operations of 8 GPT models.

Model Name	n_{layer}	d_{model}	n_{head}	d_{head}	Params(M)	FLOPs(M)
GPT2-small	12	768	12	64	81	180
GPT2-medium	24	1024	16	64	288	624
GPT2-large	36	1280	20	64	675	1440
GPT2-XL	48	1600	25	64	1406	2963
GPT3-small	12	768	12	64	81	180
GPT3-medium	24	1024	16	64	288	624
GPT3-large	24	1526	16	96	648	1368
GPT3-XL	24	2048	16	128	1152	2400

Notes: The input embedding layer is not included in the number of parameters and FLOPs.

We select NVIDIA T4 as the GPU benchmark as it also uses GDDR6 as memory for a fair comparison. For GPU, latency is recorded using `torch.cuda.Event()`, and power is measured with `pynvml`, which is a wrapper around the NVIDIA management library. The dynamic power consumption is tracked at each token generation and multiplied with the corresponding latency to get the total energy. For CPU characterization, we use python package `time.time()` for latency measurement and an open-source terminal tool `s-tui` for power monitor. In each measurement, we generate 1024 tokens and repeat 10 times, and report average energy and latency values.

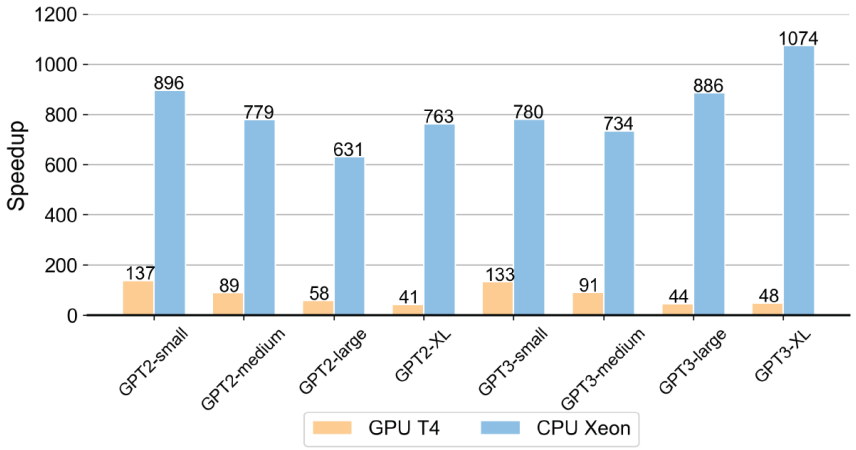


Figure 3.13: Speedup w.r.t. GPU and CPU.

3.4.2 Overall System Performance

As shown in Figure 3.13, the PIM-GPT system achieves remarkable performance improvements over GPU and CPU: 41−137× speedup over GPU T4 and 639−1074× speedup over CPU Xeon for the 8 GPT models. The high speedup originates from three aspects: 1) Memory bottleneck is effectively removed by performing the memory-intensive VMM operations inside PIM channel; 2) The mapping strategy maximizes computation parallelism and data locality; 3) Different workloads are efficiently distributed between PIM and ASIC. In comparison, GPU is not suitable for sequential token generation, since the large memory footprint and low data reuse rate under-utilize the GPU computation resources [108].

We also compare the PIM-GPT performance with previously reported Transformer accelerators, as shown in Table 3.3. SpAtten [107] accelerates GPT2-medium by 35× over GPU, and TransPIM [20] obtains similar speedup of 33×. Both ignore the layer normalization and residual connections in Transformer models. DFX [108] provides 3.2× latency reduction on average. PIM-GPT achieves state-of-the-art performance with 89× speedup over GPU. We also want to highlight that the PIM-GPT testing result is based on 1024 tokens, which cannot be supported by these prior prototypes.

Figure 3.14 shows that the PIM-GPT system achieves energy reduction of 123−383× and 320−602× over GPU and CPU, respectively. PIM-GPT effectively eliminates the energy con-

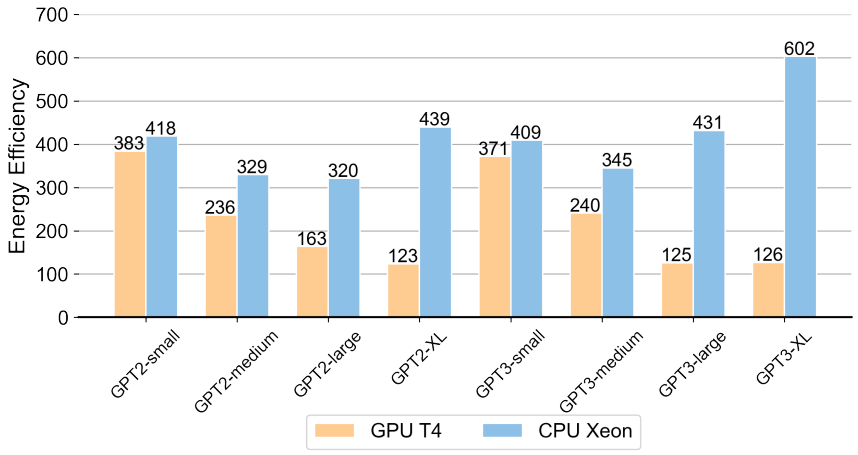


Figure 3.14: Energy efficiency improvement w.r.t. GPU and CPU.

sumption of DRAM data transmission by using PIM to locally consume data. Additionally, the mapping method leverages data locality and minimizes the row ACT and PRE operations that are energy consuming. The ASIC only contributes a very small fraction of the total system energy, but provides highly efficient arithmetic computations.

In comparison, DFX only achieves 3.99× higher energy efficiency compared to the GPU baseline [108]. TransPIM reports ~250× energy reduction [20]. SpAtten reports 382× over GPU [107], but only the attention layer is considered for energy consumption while others including FFN layers are not considered. Speedup and energy efficiency for the above mentioned accelerators are summarized in Table 3.3.

3.4.3 Detailed Performance Analysis

The layerwise latency breakdown of GPT3-small and GPT3-XL in Figure 3.15 shows that VMM operations dominate the total execution time for PIM-GPT. All other arithmetic computations only account for 1.16% of total latency in GPT3-XL. For larger Transformer models, the improvement of PIM-GPT over GPU is reduced. This is because larger GPT models allow better utilization of GPU computation resources, while PIM has limited computation resources and the performance of PIM-GPT is computation-bounded. As a result, the gain over GPU is reduced when compared

Table 3.3: Comparison with other GPT accelerators.

	SpAtten [107]	TransPIM [20]	DFX [108]	**PIM-GPT** [32]
Memory	HBM	HBM	HBM+DDR4	GDDR6
End-to-end	✗	✗	✓	✓
PIM	✗	✓	✗	✓
Data Type	INT	INT	FP16	BF16
Largest Model	GPT2-medium	GPT2-medium	GPT2-XL	GPT2/3-XL
Longest Token	32	-	128	8096
Speedup	35×	33×	3.2×	89×
Energy Efficiency	-	~250×	3.99×	221×

Notes: Speedup and energy efficiency are over GPU. PIM-GPT results are based on 1024 token generation. Other architectures generate the specified token lengths.

with smaller GPT models, although the performance gain (>40×) is still significant.

The data locality is optimized during model mapping to reduce the memory access time and enhance the computation throughput, which is evaluated by row hit rate. If a data is access from the row buffer without activating the bank row, such an access is a row hit. The row hit rate is calculated by number of hit over total data access. Figure 3.16 plots the row hit rates, achieving ~ 98% for all tested GPT models.

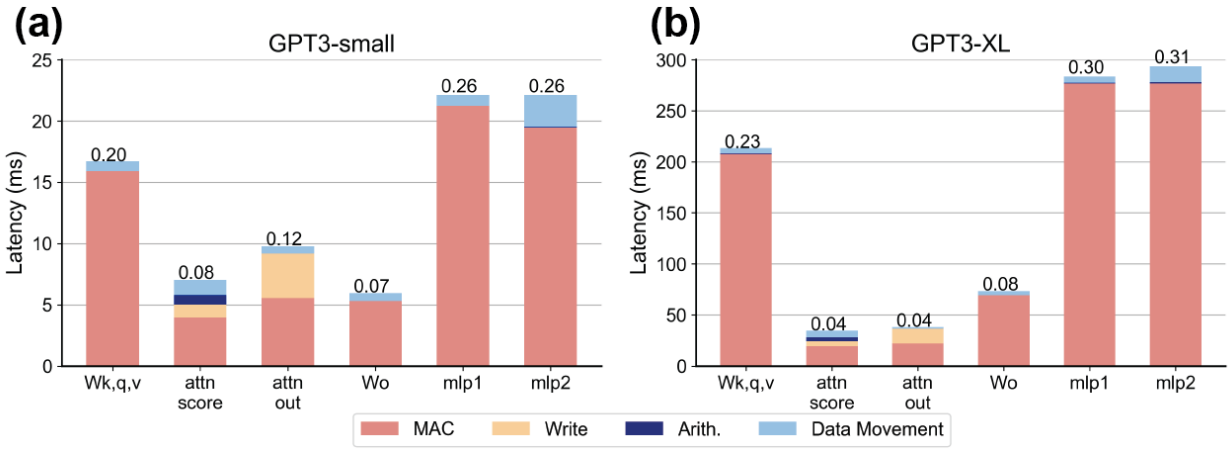


Figure 3.15: Layer-wise latency breakdown of (a) GPT3-small and (b) GPT3-XL.

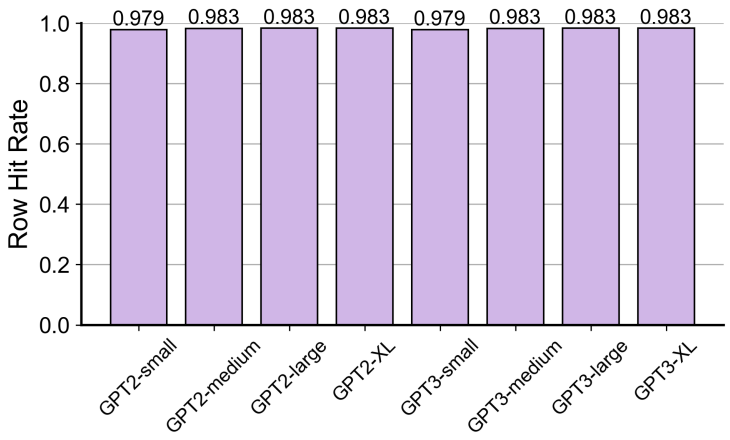


Figure 3.16: Row hit rate of bank access for 8 PGT models.

For PIM-GPT, the VMM computation occurs on the same DRAM chip that stores the required weights and Key, Value mactrices. Therefore, a significant amount of data movement can be eliminated. The data movement only happens when VMM results are transmitted to the ASIC for downstream processing or data synchronization. Figure 3.17(a) shows the data transfer reduction can be as large as 110−259×. In PIM-GPT, data movement no longer becomes the bottleneck and consumes a very small proportion of the total latency, as illustrated by the layerwise breakdown in Figure 3.15. We also evaluated the reduction of energy consumption of DRAM I/O compared to HBM2, which is commenly used in the state-of-the-art GPUs for large language models. The HBM2 access energy 3.9 pJ/bit is adopted from [120]. PIM-GPT achieves 78−184× I/O energy reduction by eliminating external matrix data accessing, as shown in Figure 3.17(b).

Since weight matrices reside in DRAM, only the input and output vectors will be transferred though the I/O. Hence, the memory access complexity can be reduced from $O(n^2)$ to $O(n)$. As a result, the energy consumption through DRAM I/O in PIM-GPT is less then 10% of total DRAM energy consumption in PIM-GPT, as shown in DRAM energy consumption breakdown in Figure 3.18. MAC operation consumes most of the DRAM energy, since VMM is the core part of the GPT. Other DRAM operations, such as ACT, PRE and REF, along with the standby energy consumption consumes more the around 30% of DRAM energy.

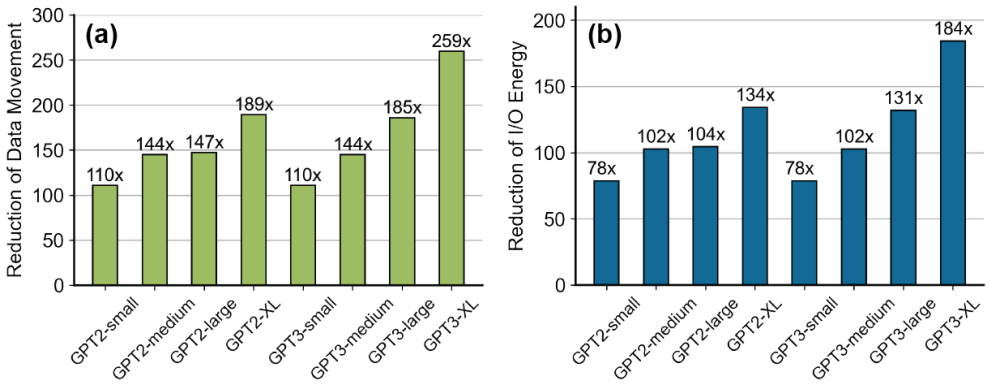


Figure 3.17: (a) Reduction of data movement. (b) Reduction of DRAM I/O energy consumption.

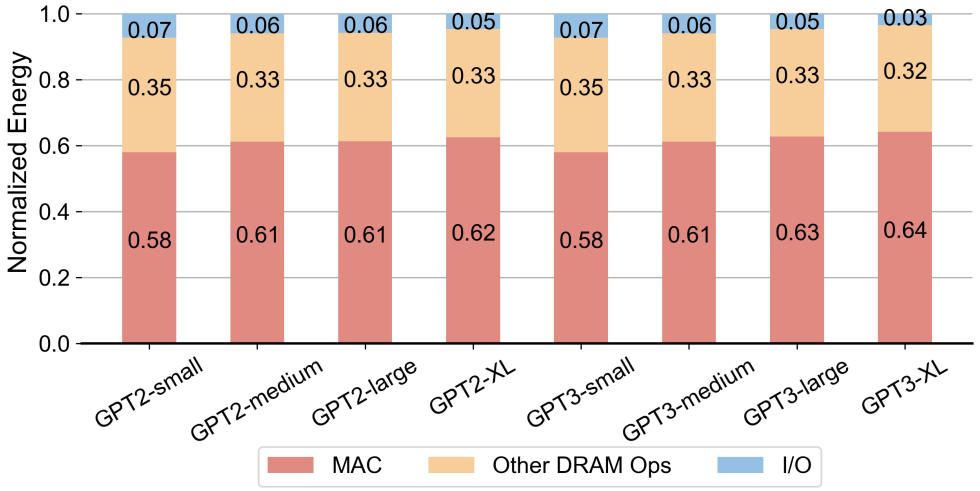


Figure 3.18: DRAM energy consumption breakdown of all models.

3.4.4 Sensitivity Study

We conduct sensitivity studies on the ASIC clock frequency and data transfer rate between PIM and ASIC to prove the robustness of the PIM-GPT design.

The PIM-GPT ASIC is designed with TSMC 28 nm technology at 1 GHz clock frequency. However, frequency scaling is an important technique to optimize the power consumption, especially for edge devices. We conduct a sensitivity study of ASIC frequency by varying the latency setting in our simulator. Figure 3.19 shows the latency at different clock frequencies for the 8 GPT models, where the latency is normalized with respect to the 1 GHz results. Overall there is only

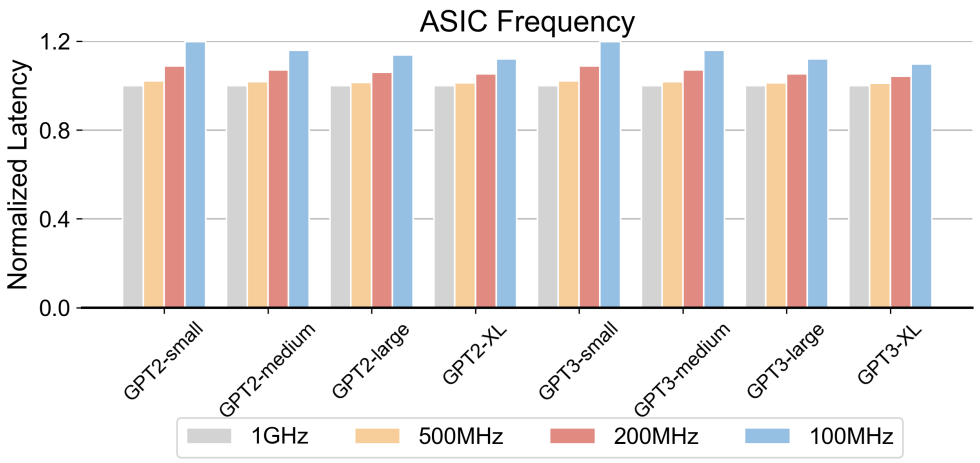


Figure 3.19: Sensitivity of performance to ASIC clock frequency.

a small latency increase when the ASIC frequency scales down from 1 GHz to 200 MHz for all models. Even when further scaling down the frequency to 100 MHz, which is 10 times slower than the baseline, the worst case only incurs a performance slowdown of 20%. Moreover, the larger models are less sensitive to the ASIC frequency scaling, since their operations are more dominated by VMM and the proportion of ASIC arithmetic computation is less than in smaller models. Therefore, the PIM-GPT design is not sensitive to ASIC clock frequency, which justifies its use for edge applications where power often needs to be optimized by reducing clock frequencies.

The memory interface can be a bottleneck for many computation tasks, even for PIM implementations. In [127], SK Hynix reported accelerating fully connected layers in GPT models using GDDDR6-AiM system. The system with 4 channels experienced ~3× slowdown when memory interface bandwidth changes from 16 Gb/s/pin to 2 Gb/s/pin. We test the PIM-GPT's sensitivity to the memory interface by changing the bandwidth configuration in our simulator. Figure 3.20 shows the latency as a function of memory interface bandwidth for 8 GPT models. When the memory interface bandwidth changes from 16 Gb/s/pin to 2 Gb/s/pin, the end-to-end GPT inference time is increased ~1.5× on average. That is 2× better than reported in [127], where only VMM and GELU in GPT are considered. Even when the data transfer rate is decreased to 1 Gb/s, all models are slowed down by ~2× on average. These results show PIM-GPT is not sensitive to the memory interface bandwidth, since most data are consumed locally and the data transfer requirements are

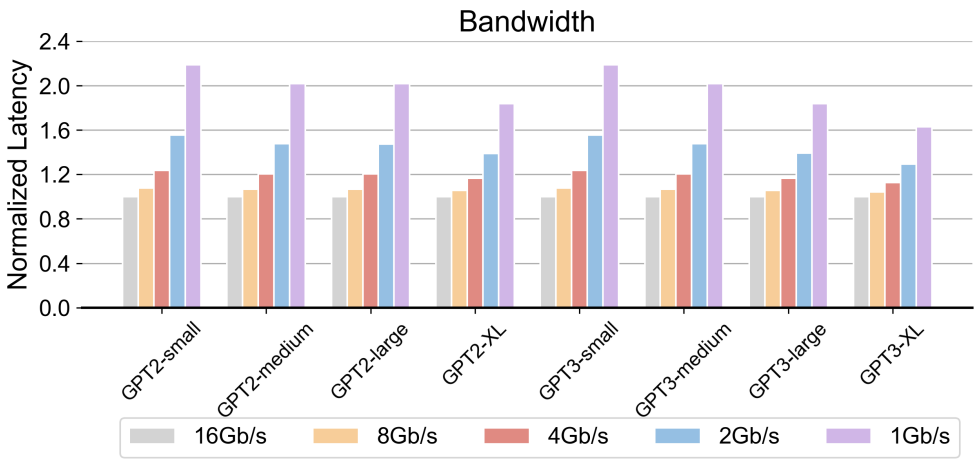


Figure 3.20: Sensitivity of performance to data transfer rate.

significantly reduced.

3.4.5 Scalability

As Transformer model sizes and token lengths increase, the required computation grows quadratically with the sequence length [128]. A lot of efforts have been devoted to reduce the computation and memory footprint for longer sequences by exploiting the sparsity and quantization of Transformer models [107] [129][130]. Another approach is to approximate the softmax attention, e.g. linear Transformer [128], Performer [131] and spikeGPT [132], which reduces the space complexity of attention from quadratic to linear. However, approximations can impact the accuracy of large Transformer models, while dynamic pruning at runtime adds hardware design complexity [107][122]. The proposed PIM-GPT mapping scheme evenly distributes model parameters across PIM channels, therefore can support more than 8k token generation for GPT3-XL. Figure 3.21 shows normalized latency of different token lengths with respect to 1k tokens.

Since the performance of PIM-GPT is computation-bounded, the latency can be reduced by adding more computation resources. Figure 3.22 evaluates the scalability of the system. When the computation throughput of the MAC unit is increased from processing 16 MACs to 64 MACs at a time, the latency is reduced by 1.8× and 2.0× for GPT3-small and GPT3-XL, respectively, as shown

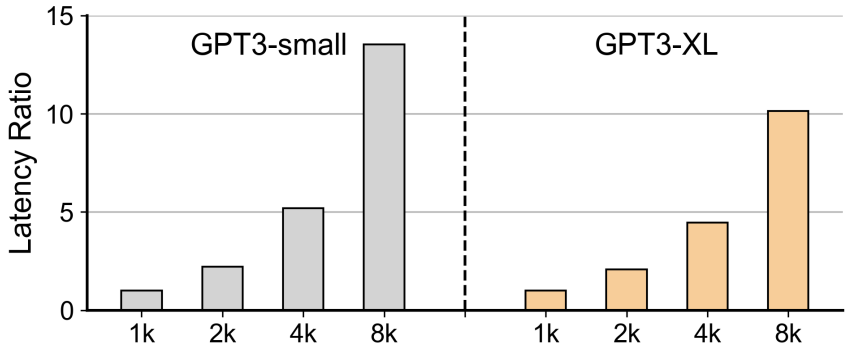


Figure 3.21: PIM-GPT latency with increased token length.

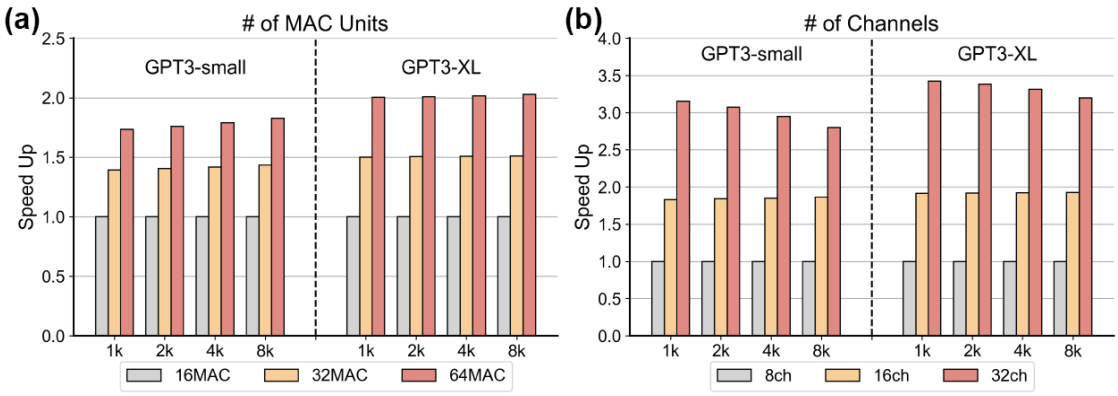


Figure 3.22: Scalability of PIM-GPT with increased number of (a) MAC units and (b) channels.

in Figure 3.22(a). The speedup is sub-linear. This is because when PIM performs MAC operations on a bank row, the row needs to be activated and precharged, which consumes a significant amount of time. We also evaluate performance enhancement by increasing computation parallelism, as shown in Figure 3.22(b). More memory channels can be attached to the ASIC with relatively minor modifications to the ASIC data port. The improvement scales almost linearly with the number of channels. The speedup slightly reduces for longer sequences, because longer sequence requires more arithmetic computation for ASIC to process.

3.5 Conclusion

In this work, we propose a hybrid hardware system, PIM-GPT, to accelerate the memory-bounded GPT token generation tasks. PIM-GPT consists of GDDR6-based PIM chips and a lightweight ASIC chip to support end-to-end GPT acceleration. Model mapping and workload distribution are optimized to maximize computation parallelism and data locality. The proposed system achieves $41-137\times$, $631-1074\times$ speedup and $123-383\times$ and $320-602\times$ energy efficiency over GPU and CPU on 8 GPT models. We highlight that the design only requires light modifications to the DRAM architecture, which offers a practical solution for edge applications. PIM-GPT exhibits other favorable characteristics, including low sensitivity to ASIC clock frequency and memory interface specs, capable of long token generation and excellent scalability.

CHAPTER 4

Model Extraction Attack on PIM Architectures by Side-Channel Analysis

4.1 Background and Motivation

PIM architectures can circumvent the von Neumann's bottleneck when accelerating communication-limited tasks, such as deep learning workloads [75][76][133]. However, the security vulnerabilities of analog PIM architectures are yet to be evaluated, and this becomes of paramount importance when the target market of low-power PIM accelerators are in ubiquitous, edge-based computing that transmit and receive information from a variety of sensors [134] [135]. In theory, side-channel attack can be used to extract and infer information pertaining to the on-chip DNN model deployed during inference [83], as shown in Figure 4.1.

When PIM is coupled together with mixed-signal computation, as with RRAM crossbar hard macros, it is thought that analog computation via bit-line current summation or charge accumulation offers a further layer of obfuscation. Co-locating memory and processing in a tiled architecture eliminates data movement between memory and processor [4][75]. Since the weights are stationary, weight memory access can be further limited. Hence, it is more challenging for malicious users to compromise the security of these hardware accelerators. Conventional DNN accelerators, such as GPUs, use single-instruction-multiple-threads/data (SIMT/SIMD) execution and must therefore time multiplex operations that take place across different DNN layers. Rich data-dependent information, such as read/write volume, memory address track, and execution latency, can be obtained

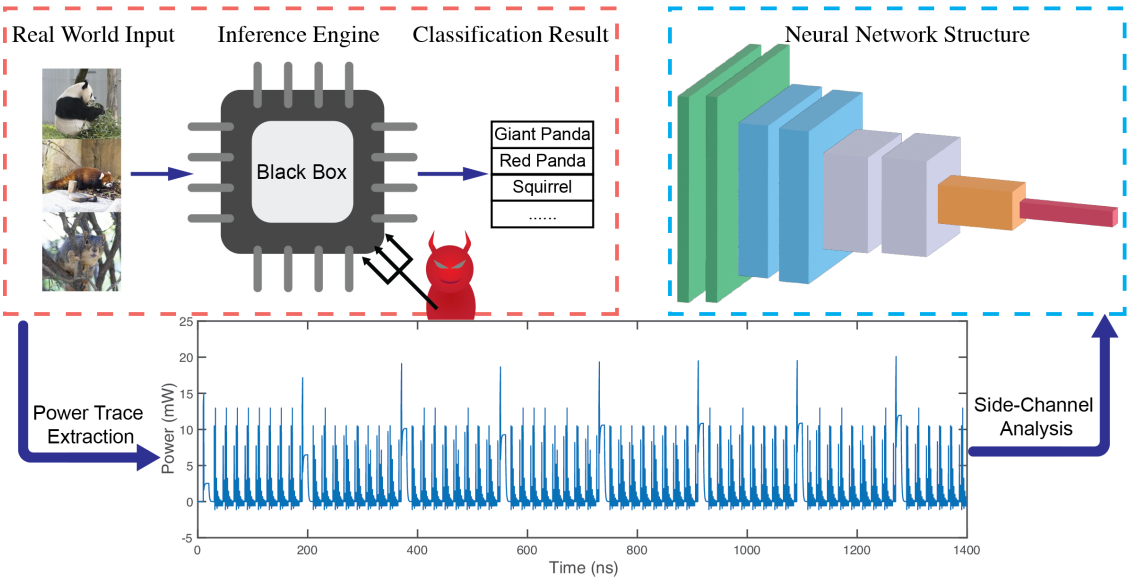


Figure 4.1: Schematic of side-channel attack on PIM chip.

from a bus-snoop attack or side-channel attack [79]. As DNN models are generally highly proprietary, the neural network architectures become valuable targets for attacks. In PIM systems, since the whole model is mapped on chip and weight memory read can be restricted, the pre-mapped DNN model acts as a "black box" for users. However, the localized and stationary weight and data patterns may subject PIM systems to other attacks.

In this chapter, we demonstrate that the complete network architecture for DNN models stored inside the isolated memory blocks in a PIM system may be extracted from the power trace of each tile, without prior knowledge of the model [33]. Our side-channel attack was performed by simulating the dynamic power trace of the mixed-signal RRAM PIM macros. By analyzing the power trace of the different PIM macros, the layer type and sequence, output channel/feature size of convolutional/fully connected layers as well as kernel size of convolutional layers, can be inferred. As an example, we demonstrate how we are able to reverse engineer the full LeNet [136] architecture from a mixed-signal RRAM accelerator by side-channel analysis, without prior additional knowledge of the neural network model in use. We also propose several countermeasures that can potentially make the PIM systems resistant to such side-channel attacks, which should be considered during hardware and compiler design.

4.2 Simulation Framework

4.2.1 Overview

Modern PIM applications such as DNNs for machine learning may require a large number of RRAM PIM tiles, making it impractical to perform analog SPICE simulations that can account for all parasitics [137]. Instead, the hardware performance is often estimated through simulators that integrate simplified device and circuit models to produce inference accuracy, hardware power and area, with rapid iteration times.

Architecture-level simulators such as MNSIM [93] and NeuroSim [91] provide a flexible interface for a wide range of design options, and can effectively simulate hardware performance (e.g. power, area, and latency) based on simulator-embedded circuit-level models. However, these works, amongst other simulators [138] [139] [140] [141] [92], mainly focus on inference accuracy and global hardware performance. Time dependent data, e.g. dynamic power traces, are less explored. As we will demonstrate, dynamic power and other related time-varying information that is available from read-out ports of RRAM macros can prove to be useful in performing side-channel attack, and thus expose an underexplored vulnerability in mixed-signal PIM accelerators. Furthermore, dynamic power information is also helpful for thermal aware optimization [142] [143].

To analyze the dynamic power information of PIM systems, we first developed a mixed-signal power simulator. The overall simulator framework is shown in Figure 4.2. The simulator offers two interfaces: i) one during configuration that allows users to define hardware-level properties (system configurations), and perform mapping of a pre-trained neural network model through a PyTorch [144] interface, and ii) another during runtime simulation, which takes a dataset as an input. The pre-trained weights are mapped to RRAM conductance values based on weight and device precision, as well as the permissible device conductance range, following standard approaches [133] [145]. Each feature (or pixel) of the input data samples, e.g., images, is scaled and converted to a bit-serial input, across a given number of clock cycles determined by the specified input precision.

Different from prior efforts that aimed to simulate the hardware performance as a whole, or

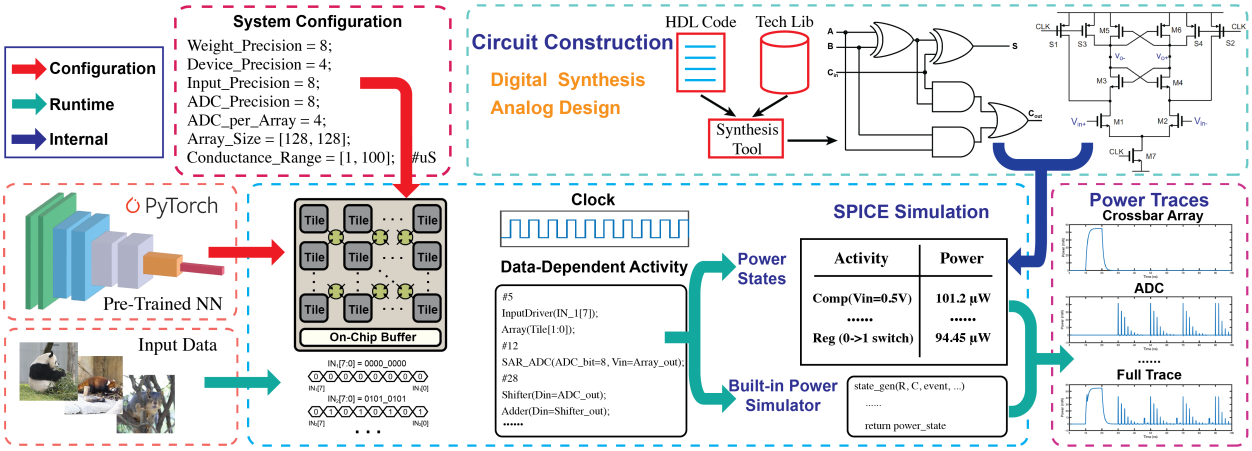


Figure 4.2: Dynamic power simulator framework.

the performance of each component at runtime, we target the time dependent power data that can be used to reverse engineer the tasks being performed in the PIM systems. The simulator is designed to provide reliable dynamic power information based on the input and memory data patterns, while enabling rapid experimental iterations. In particular, the dynamic switching power is calculated at each clock cycle, since the transition power is data dependent and will be critical for power trace analysis [146]. To provide reliable power simulations, we synthesized the deployed digital components (e.g. adder, register) and custom-designed analog components (e.g. ADC, MUX) using a TSMC 28 nm technology. Complex digital circuit components were modeled by sub-dividing them into basic units that can process 1 bit data, where high-to-low and low-to-high transition powers are extracted based on the technology database. For analog components such as the comparator in the ADC, the dynamic power of each input voltage is recorded at 100 mV intervals. The recorded power data is used to generate a built-in lookup table (LUT). The circuit modules subsequently refer to the LUT and generate their power transition states. For simpler resistive and capacitive circuits, such as RRAM and capacitive digital-to-analog converter (CDAC) in successive approximation registers (SAR) ADC, we developed a built-in power simulator to compute their power traces. Each circuit component will generate its own power trace which is subsequently merged as the full trace for a given tile.

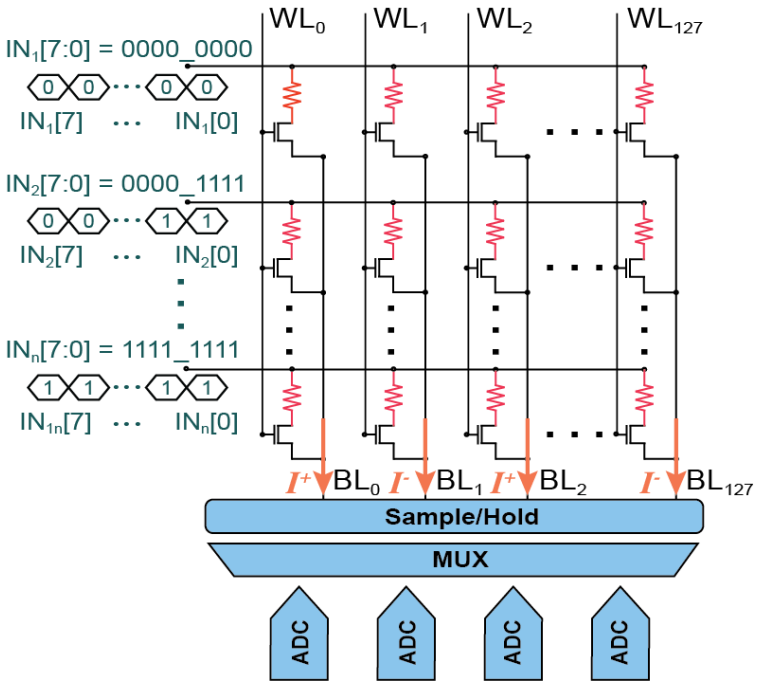


Figure 4.3: RRAM tile for VMM.

4.2.2 RRAM Array and ADC

Schematic of the RRAM array model are provided in Figure 4.3. During the inference phase, the word-line drivers turn on the select transistors in the 1T1R structure. The input data are converted into voltage pulses in bit-serial fashion, and applied to each source-line. The currents through the RRAM devices are accumulated along the vertical bit-lines. We include the effect of the parasitic capacitance seen from the bit-line which introduces a propagation delay during the inference phase. To account for positive and negative weights, we implement a current mirror-based subtractor circuit, which subtracts the output current from the positive and negative weight columns. The subtracted current will charge or discharge a sampling capacitor, which has been pre-charged to VDD/2. The capacitor will hold the analog voltage for the ADC, before the outputs are digitized.

Figure 4.4(a) shows the schematic of a synthesized 8-bit charge redistribution SAR ADC design in the simulation, and Figure 4.4(c) depicts its timing diagram. During the sampling phase, the switch S is closed and all capacitors are connected to V_{in}. Next, the switch S is opened and all capacitors are grounded via their bottom terminals, raising the voltage at the positive input terminal

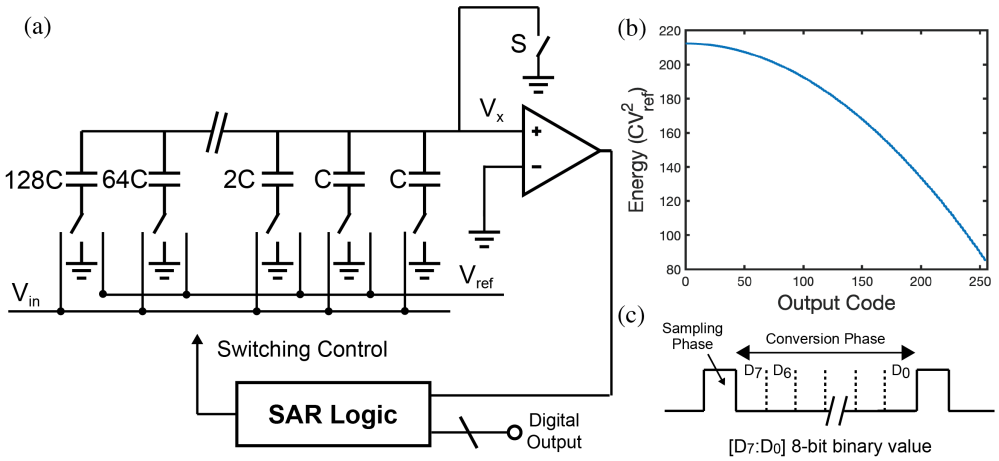


Figure 4.4: (a) Schematic of an 8-bit SAR ADC. (b) DAC switching energy with respect to output codes. (c) Timing diagram of the SAR ADC.

of the comparator to $-V_{in}$. During the conversion phase, SAR logic controls the switches to V_{ref} one by one to perform a binary search. The conversion takes 8 cycles and the energy consumption of the n-th step can be calculated from the change in V_x, and the capacitance connected to V_{ref} using Equation 4.1.

$$E = \begin{cases} -C_1 V_{ref}(\Delta V_x - V_{ref}), & n = 1 \\ -V_{ref}(\Delta V_x \sum_{i=1}^{n-1} C_i D_i + C_n(\Delta V_x - V_{ref})), & n \neq 1 \end{cases} \tag{4.1}$$

where D_i is the i th MSB output code. When $D_i = 1$, the i-th switch connects to V_{ref}, otherwise, it connects to ground.

The dynamic power from the transitions in the DAC is the dominant contribution to the total power consumed by the SAR ADC [147] [148]. Figure 4.4(b) shows the switching energy with respect to the output code, and it has a clear data pattern dependency. In our simulator setup, the unit capacitor in the DAC array is set as 1 fF, and each capacitor can be charged and settled within a worst-case interval of 20 ps.

4.2.3 Digital Components

The digital components of the PIM system include input and output registers, shifters and adders within a tile, as well as inter layer logic functions such as ReLU activations, pooling operators, and routing circuits. As mentioned above, the transition power of digital components are based on a built-in LUT which stores the extracted transition power data of the basic digital units from the TSMC 28 nm technology process. To fully simulate the data flow in the hardware, all data movement between circuit modules in our simulator are performed in a binary, time-multiplexed fashion. Hence, the total dynamic power of each switching event from all digital components can be calculated based on each input bit of data.

4.2.4 Model Mapping

For inference tasks, the pre-trained weights are mapped across tiled RRAM arrays, and remain stationary during operations. To benefit from inference efficiency, the weights and input activations are quantized to 8 bits. It has been demonstrated that 8-bit weight precision does not incur a significant accuracy degradation, especially when coupled with quantization-aware training techniques [27][149] [150]. However, RRAM cells do not have sufficient internal precision to support 8-bit weights, which requires reliable programming of 256 conductance levels. As a result, it is often more practical to map a given weight across multiple cells, e.g. using 2 cells each offering 4 bits. Figure 4.5(a) shows the mapping approach used in our simulation for the convolution kernels. Other mapping strategies can be employed similarly through the configuration interface shown in Figure 4.2. Each kernel is flattened to a 1D vector, followed by splitting positive and negative values across columns. Next, each weight is quantized to 8 bits, mapped between the 4 MSBs and 4 LSBs separately. For a PIM system with 8-bit weight precision and 4-bit device precision, the total column number is thus 4 times that of the output channel size (2 for positive and negative weights, and 2 for splitting the 8-bit weights into 2 cells), and the number of mapped rows is the flattened kernel size K^2C_{in}, where K is a single kernel dimension and C_{in} is the input channel size.

The output of the convolutional layer is computed by sliding the input activation across the

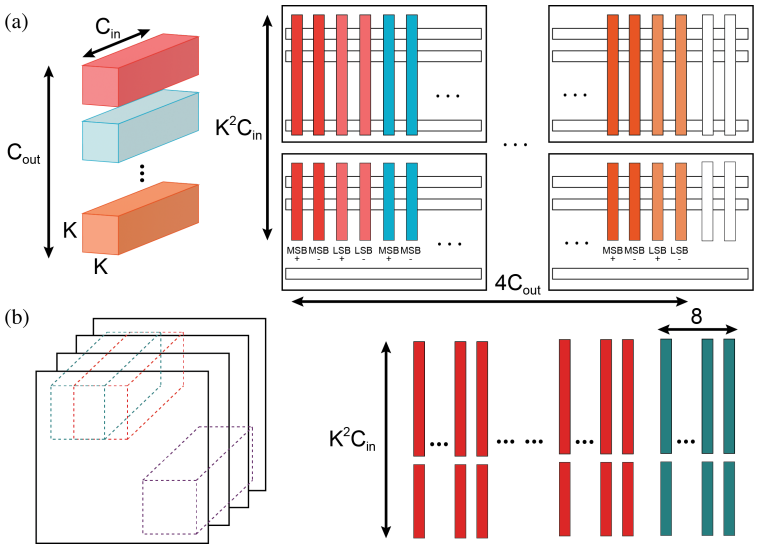


Figure 4.5: (a) Convolution kernel mapping. (b) Input data mapping.

kernels, as illustrated in Figure 4.5(b). The input to each kernel is flattened to match the kernel, and outputs from all kernels are computed simultaneously through the PIM module outputs. For each 8-bit input, we employ bit-serial representation and it takes 8 consecutive steps to compute 1 input.

The weights and inputs to a fully connected layer can be quantized and mapped directly without any reshaping. Similarly, vector outputs of the fully connected layer can be acquired simultaneously through the PIM outputs.

4.3 Experiment and Analysis

4.3.1 Experimental Setup

After model mapping, we simulate the DNN inference in the PIM system at the circuit-level, and use the extracted power traces to analyze the feasibility of model extraction attacks through the side-channel leakage. Side-channel attack analysis is based on physical phenomena during execution, as well as mathematical analysis. By carefully measuring and analyzing the power dissipation of the chip, attackers may be able to reverse engineer sensitive data or architectural information [82]. A

general assumption for the proposed attack is that the attacker already knows hardware parameters of the PIM tiles, i.e., the RRAM array size, the ADC type and the number of ADCs per tile, but has no knowledge about the mapped NN. This is a realistic assumption since memory access to the stored weights in PIM systems can be restricted and physically separated from other programs the attacker may gain access to. The attacker, as a user, however has access to the input and output ports of the chip, i.e., can control the input data supplied to the PIM system during runtime. For example, most of inference tasks of RRAM-based PIM system are pipelined during processing to improve the throughput [75] [145] [151], i.e., starting to process the second input as the first input passes through the first layer. This will blur valuable power and timing information of a single runtime. However, as the attacker has the full control of the input and output data ports, she/he can halt the next input to the system until the previous inference completion to gain more accurate power measurements.

Side-channel attack can be classified into invasive and non-invasive attacks [152] [153], based on whether decapsulation of the chip is used. Carefully designed invasive attacks make it possible to measure the power trace of each single RRAM tile on chip. In our experiment, the assumption is the attacker can only measure the power traces of each tile for side-channel attack, and does not have access to individual RRAM devices. Figure 4.6 illustrates a simplified flow for model extraction attack, which will be discussed in the following sections.

We first initialize the hardware design with the system configuration parameters shown in Figure 4.2. The conductance range of the RRAM devices is set between 1 μS and 100 μS. Each cell has 16 conductance levels. A 50 MHz clock is used to drive the bit-serial inputs, limited by the RRAM read operation speed. The other digital logic is assumed to operate at 500 MHz. Before inference, the pre-trained neural network model is mapped to RRAM arrays through the interface between the simulator and PyTorch, without any sensitive information leakage. The model is deemed 'inaccessible' to a user once the model is compiled onto the PIM accelerator, for the entirety of the side-channel attack. At the very end of the experiment, we validate the extracted model structure through side-channel attack with the ground truth.

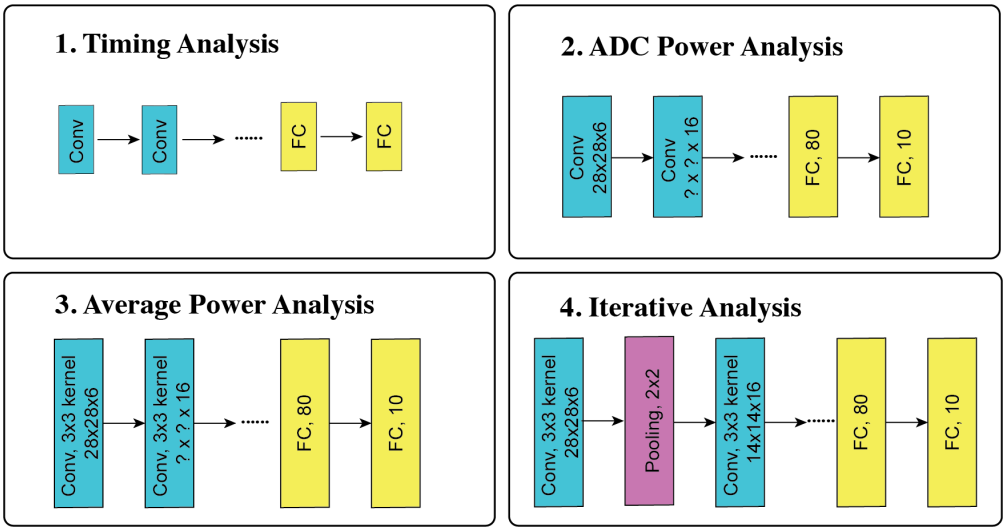


Figure 4.6: Overview of side-channel attack flow for DNN model extraction.

In this experiment we use CIFAR-10 [154] as the dataset during inference, which consists of a set of 60,000 natural color images that are 32×32 in size.

Before elaborating attack approach, we summarized the assumption for the attacker's knowledge as follows.

- The attacker knows the hardware implementation details of a PIM tile, including array size, ADC type and number of ADCs per tile. However, the attacker has no knowledge of the neural network mapped on chip.
- The attacker has the full control of the input and output ports of the chip. Hence, the next input can be halted until the completion of the previous input, allowing the attacker to control the input pipeline to get more accurate power trace of each inference.
- The attacker has no access to individual RRAM cells, but can measure the power trace of each tile as a whole.

4.3.2 Power Traces

We test the side-channel attack on the PIM system that has an unknown neural network already mapped on chip. Based on the power traces extracted from the simulator, it is found that 23 PIM

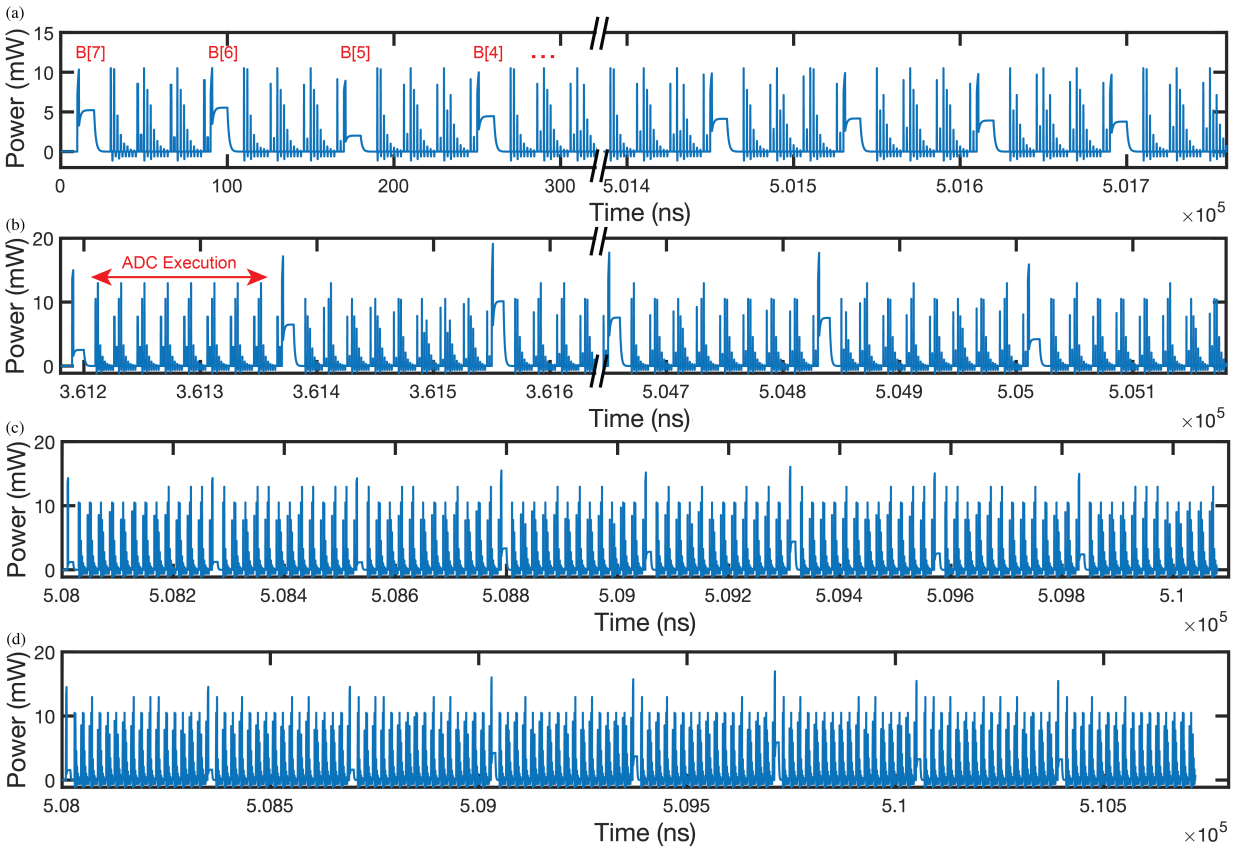


Figure 4.7: Simulated output power traces from 4 different PIM tiles during one inference task.

tiles are utilized to store the pre-trained model. Figure 4.7 displays the power traces of 4 of those tiles during an inference. Examining the power traces, we can first identify when a bit-serial input is applied to the RRAM array. As can be seen in Figure 4.7, processing a 1-bit input in the PIM module is accompanied by the characteristically transient IR power draw from the array, followed by a stable period during data passing the RRAM cells, and a switching-dense period for data conversion in the ADC along with the execution of other digital components. By inspecting the power traces, the following hardware hints can be extracted.

- **Start time:** corresponding to when a tile starts execution. Tiles with identical start times are expected to belong to the same neural network layer (e.g. Figure 4.7(c) and (d)). Grouping tiles with the same start time will provide information on the layer size and the number of layers that perform VMMs.

- **Execution time:** corresponding to how many bit-serial inputs are executed at the layer for a given inference. The number of input bits sent to a module can be inferred by identifying the arrival of a new input bit, which has the distinguishing feature of the transient IR power draw followed by a stable period of a few nanoseconds corresponding to data passing through the RRAM cells. Figure 4.7(a) illustrates this by including labels for the bit-serial inputs of the first data sample. The execution time of convolutional (Conv) layers are typically much longer than that of fully connected (FC) layers due to the larger number of computational cycles required in convolutions.

- **ADC execution time:** corresponding to how long it takes for the ADC to convert all analog outputs from the RRAM array. ADC execution time can be extracted by inspecting the dynamic power consumption of the ADC after data has passed the RRAM cells. During ADC conversion, the successive approximation steps lead to decrementing voltage changes, and the overall tendency of ADC transition power during a single conversion is also decreasing. The ADC execution time indicates how many columns are being utilized in an array.

- **Average Power:** $\frac{1}{T}\int_{T_1}^{T_2} P(t)dt$. For arrays with the same number of utilized columns, the ratio of average power is statistically proportional to the number utilized RRAM rows.

Using these features, we will show how to extract the full neural network architecture in Section 4.4.

4.4 Neural Network Extraction

4.4.1 Layer Property Extraction

The number of neural network layers that perform VMM operations and their corresponding layer types can be identified first, based on the start time and execution duration. The extraction algorithm of these layer properties is summarized in Figure 4.8. The algorithm first extracts the start time and execution time of each tile. Whether a tile belongs to a Conv layer or a FC layer

Algorithm 1: Layer Property Extraction

Data: Power traces of all tiles
Result: A list of NN layers that perform VMM operations

```
for i = 1 to TileNum do
    Tile[i].StartTime ← StartTimeExtract(trace[i]);
    Tile[i].VMMExTime ← ExTimeExtract(trace[i]);
    if Tile[i].VMMExTime == 1 then
        Tile[i].Type = fc;
    else
        Tile[i].Type = conv;
    end
end
LayerSequence = GroupNSortTiles(Tile);
```

Figure 4.8: Layer property extraction algorithm.

can then be identified based on how many VMM operations are executed. The number of VMM operations for a Conv layer corresponds to the output feature map size, but is fixed at '1' for a FC layer. Finally, tiles with the same start time are grouped, and the layer sequence is generated by sorting the start time of these grouped tiles.

In our experimental test case, using this approach, the 23 tiles utilized in the network inference were found to belong to 2 Conv layers and 3 FC layers.

4.4.2 Output Channel Size Extraction

Based on the mapping approach in Section 4.2.4, we know that the output channel size of a given Conv layer and the output feature size of a FC layer are directly related to the number of the utilized RRAM columns in the PIM modules. The number of utilized columns in each module can be extracted from the ADC execution time. In the system design we analyzed, each tile has 128 columns and generates 64 output currents, where the subtraction of positive and negative bit line currents is performed in the analog domain. Each tile also has 4 ADCs. Hence, each of the 4 ADCs will execute 16 times to convert the 64 outputs for a fully mapped array. In a partially

mapped tile, shorter ADC execution time will be observed, so the ADC execution time can be utilized to estimate the output channel size. The algorithm is summarized in Figure 4.9. We first extract the ADC execution time from the power traces. If the ADC execution time is smaller than the maximum execution time, which is 16 in our case, it will be labeled as a column-wise partially mapped tile (i.e., the array is not fully utilized). The number of partially mapped tiles and fully mapped tiles are counted, which then allows us to calculate how many columns are mapped and the output channel size for the given layer.

Algorithm 2: Output Channel Size Extraction

```
Data: Power traces of tiles in one layer
Result: Output channel/feature size of Conv/FC
        layer
for i = 1 to TileNum do
    Tile[i].ADCExTime ←
     ADCExTimeExtract(trace[i]);
    if Tile[i].ADCExTime < ColNum/(2× ADCNum)
     then
        ColPartialMapTile++;
        ColNum = Tile[i].ADCExTime×2×ADCNum;
    end
end
FullCol = TileNum / ColPartialMapTile - 1;
OutSize = (FullCol × ArrayCol +ColNum)/
  CellPerWeight;
```

Figure 4.9: Output channel size extraction algorithm.

As an example, Figure 4.10 plots the extracted utilization and power traces identified from the first FC layer in the unknown neural network model. From the algorithm, we can conclude there are 4 column-wise partially mapped tiles. The mapping information of the 16 tiles used to map the FC layer is shown in Figure 4.10(a). Figures 4.10(b) and (c) display the ADC execution time traces for the two tiles marked in (a) for a given input. From inspection, the ADCs in the tile marked by the red line executed 16 times, indicating it is fully mapped and utilized. The ADCs from the tile marked by the blue line only executed 12 times, indicating 96 columns are mapped.

By analyzing the power traces from the 16 tiles, we can calculate the output feature size to be $(3 \times 128 + 96)/4 = 120$.

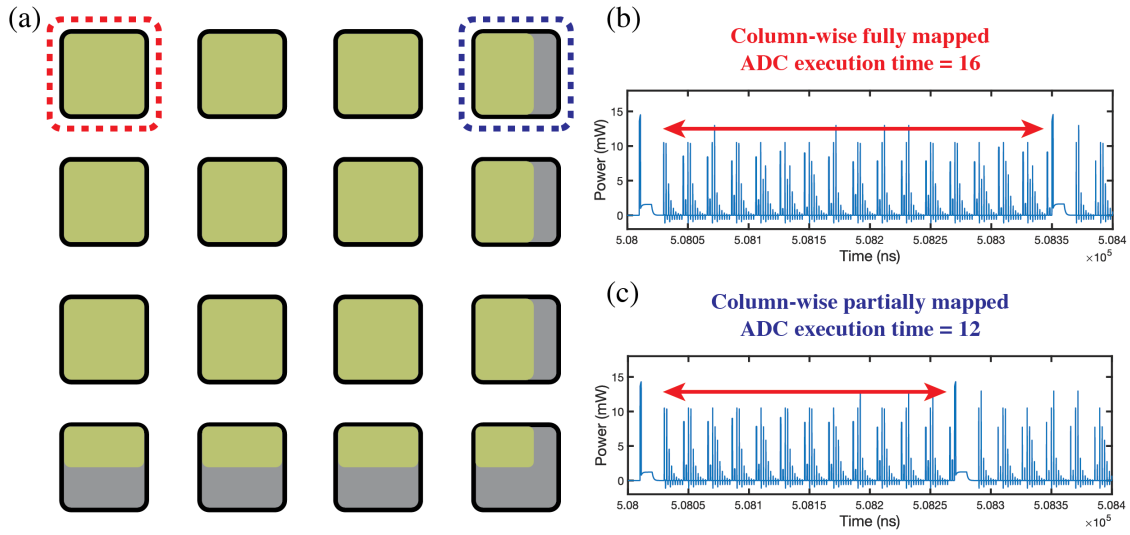


Figure 4.10: (a) PIM tiles mapped with weights from the first FC layer. (b) ADC execution time of a column-wise fully mapped tile, and (c) a column-wise partially mapped tile.

Using the same approach, the output channel sizes of the first two identified Conv layers can be uncovered and found to be 6 and 16 from power traces in Figure 4.10 (a) and (b). The output feature size of the second identified FC layer is 84. The output feature of the third identified FC layer corresponds to the number of classification classes, and is 10 for the CIFAR10 dataset.

4.4.3 Kernel Size Extraction

The weights of the Conv kernels are mapped across K^2C_{in} rows. Therefore, there are two possible cases: K^2C_{in} is either smaller than (or equal to) the number of rows of a single RRAM tile, or it exceeds the number of rows available and must be distributed across multiple tiles. The number of input channels characteristically increases with deeper Conv layers, so the first case of a kernel fitting within a single array typically only happens with the first layer.

For the first Conv layer, the size of the input image is known to be $W_{in} \times W_{in}$. As the Conv layer computes one output pixel with one input vector, the output feature map size of the first Conv layer can be speculated from the VMM execution time in Algorithm 1. Furthermore, the input and

output feature maps follow the relationship shown in Equation 4.2:

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1, \tag{4.2}$$

where W_{out} and W_{in} are the output and input widths (equal to the heights), K is the kernel size, P and S are for padding and striding.

The first identified Conv layer was fully mapped within an array. Since the total row number is $K^2C_{in} < 128$, where C_{in} is 3 for RGB images, and K^2 must be a square number, only a few kernel sizes are possible. Thus, we can uncover the kernel size of the first Conv layer by testing the range of possible kernel sizes iteratively using Equation 4.2. Most kernels use odd-numbered dimensions which further narrows the search space to $K = 1, 3, 5$. By testing all cases and comparing with the output shape, we found the kernel size is 5×5. This approach is also helpful when analyzing the pooling layer, which will be discussed in further detail in the next subsection.

Algorithm 3: Kernel Size Extraction

Data: Power traces and input channel size of a Conv layer
Result: Kernel size of Conv layer

```
for i = 1 to TileNum do
    Tile[i].AvePower ← MeanPowerExtract(trace[i]);
end
TotalAvePower ← TileColWiseSum(Tile.AvePower);
if TileRow ≠ 1 then
    RefPower =
     Mean(TotalAvePower[Row= 1 : n − 1]);
end
TotalRow = (TileRow −1 + TotalAvePower[n] / RefPower )×
 ArrayRow;
KernelSize = round(sqrt(TotalRow/InChannel));
```

Figure 4.11: Kernel size extraction algorithm.

In most cases beyond the first Conv layer, the input activation includes many channels, and the kernel must be mapped across multiple tiles. Figure 4.12(a) shows how the identified second Conv

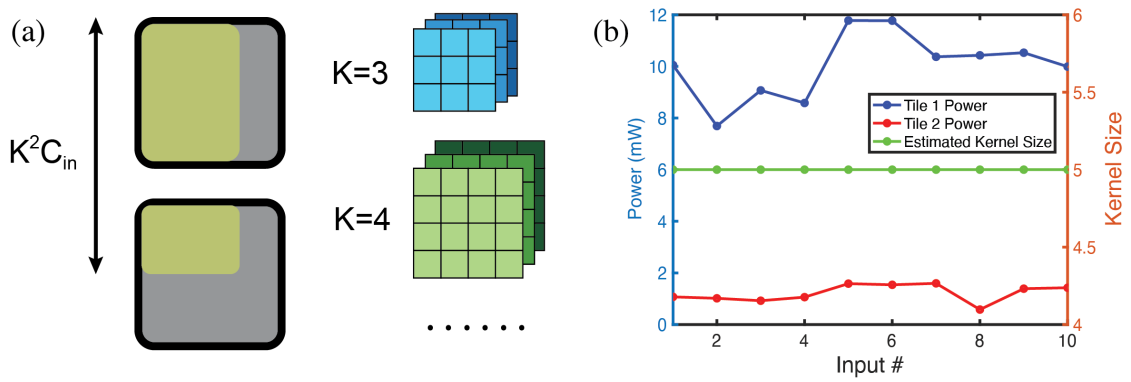


Figure 4.12: (a) Tiles mapped with weights from the identified second Conv layer and possible kernel sizes. (b) Average power and predicted kernel sizes with respect to different input images.

layer utilizes 2 RRAM tiles. For deeper kernels, the kernel size can be extracted by analyzing the average power consumption of all tiles in the Conv layer, with extraction steps summarized in algorithm in Figure 4.11. We first extract the average RRAM array power of each tile from the obtained power traces. Next, if the columns are mapped across multiple tiles like in Figure 4.10(a), the average power of each tile row is summed. The reference power is generated by calculating the average power of row-wise fully mapped tiles. The number of utilized rows is obtained from the ratio of the average power of row-wise partially and fully mapped tiles. With this approach, it is not feasible to extract exactly how many rows are mapped. However, as discussed earlier, the total row number is K^2C_{in}, and C_{in} has been extracted by Algorithm 2 from the previous layer while K^2 must be a square number. Hence, we only need to find the nearest possible square number that matches the row number estimated from Algorithm 3. Figure 4.12(b) shows the average array power of two tiles in the identified second Conv layer with respect to the input image number. While the average power varies among different inputs, the algorithm always predicts the correct kernel size due to the limited number of permissible kernel sizes.

Using the same approach, the input feature size of the first FC layer can be extracted from the input data dimension which takes the form N^2C_{out}, where C_{out} is the number of output channels in the previous Conv layer and has been extracted by Algorithm 2.

4.4.4 Pooling Layer Analysis

A 2D pooling layer is typically square-shaped, and placed between two Conv layers, or between a Conv and a FC layer. For the latter case, the pooling layer can be easily extracted since the output shape from the Conv layer and the input shape for the FC layer are both known. The ratio of these two indicates how the shape shrinks after the pooling layer and the size of the pooling operation.

Algorithm 4: Pooling Detection

Data: Output shape and number of channels of the current Conv layer. Output and kernel size of the next Conv layer, or input feature size of next FC layer.
Result: Size of Pooling layer

```
1 if NextLayer == fc then
2 |   PoolingSize = ConvOutShape /
  |     sqrt(FcInFeature/ConvOutChannel);
3 else if NextLayer == conv then
4 |   InSize ← TryPadStride(Kernel→Next,
  |     OutSize→Next) ;
5 |   PoolingSize ← TryPooling(InSize, OutSize →
  |     This) ;
6 end
```

Figure 4.13: Pooling layer detection algorithm.

However, the information of the pooling layer between two Conv layers cannot be trivially extracted as the input shape of the Conv layer is unknown. Therefore, we propose an iterative search approach summarized in Figure 4.13. With the output kernel size of the second Conv layer, we can reconstruct a series of input shapes with different padding and stride properties. As the output shape of the first Conv layer is known, we can estimate the output shape of the pooling layer based on different pooling shapes, which is then compared with the reconstructed input shape for the second Conv layer. Table 4.1 summarizes possible input shapes. Among all of them, only dimensions 14×14 and 28×28 can be supported with 2×2 pooling or without pooling, respectively. The pooling layer is implemented in digital logic, and will incur a delay in the execution start time of the next Conv layer. Hence, based on the above analysis and the delay observed in the start times,

we can speculate that a 2×2 pooling layer exists between two Conv layers.

Table 4.1: Pooling layer detection attempt

Padding	Stride	Input Size	Pooling
0	1	14×14	$\mathbf{2 \times 2}$
1	1	12×12	-
2	1	10×10	-
...	...	...	...
2	3	28×28	$\mathbf{1 \times 1}$
3	3	26×26	-
4	3	24×24	-

4.4.5 Discussion

Figure 4.14 shows the extracted neural network architecture based on the analysis above. The colored labels indicate the architectural information that has been extracted from which proposed algorithm. The extracted neural network architecture matches the ground truth – a LeNet model, using only side-channel attack without any prior knowledge of the model. The model comparison is further validated with the original PyTorch model file. We expect this approach can be generalised to any neural network model that consists of Conv, FC, and 2D pooling layers. Key results of this work and comparison with prior studies are listed in Table 4.2.

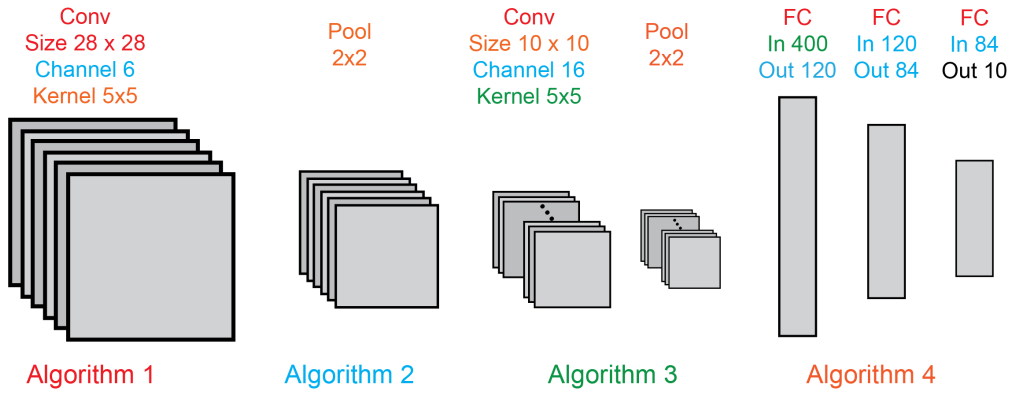


Figure 4.14: Extracted complete neural network architecture for the unknown model. The colors label the algorithm used to obtain the architecture information.

Table 4.2: Comparison of different works performing model extraction attack on DNN accelerators.

	Platform	Leaked Data	Data Acquisition	Extraction result
DAC'18 [84]	FPGA accelerator + off-chip memory	Memory and timing	Simulation	Layer features, sizes
USENIX Security'19 [85]	ARM Cortex-M3 microcontroller	electromagnetic emanation and timing	Electromagnetic probe measurement	Layer features, sizes, activation functions
ASPLOS'20 [79]	GPU	Memory access volume and timing	Monitoring PCIe and GDDR memory bus	Layer features, sizes, connections, activation functions
TCSII [83]	Raspberry Pi	Power	External power data acquisition card	DNN architecture type, parameter sparsity
TIFS [86]	FPGA accelerator	Power	Ring Oscillator Power sensor	Layer features, sizes, activation functions
This Work [33]	RRAM-based analog PIM accelerator	Power and timing	Simulation	Layer features, sizes

4.5 Feasibility Analysis

The above-mentioned side-channel attack methodology is proven capable to reverse engineer the unknown neural network architecture from the power traces. To validate our approach, real-world constraints need to be considered. In this section, we discuss potential data acquisition solutions, sampling rate requirements and circuit noise effects when attacking a real PIM system using the proposed theoretical methodology.

Two measurement techniques can be considered for these data dependent signals. One technique is using electromagnetic probes. As RRAM tiles with different weights are spatially located on the chip, electromagnetic probes with sufficient spatial resolution can measure the side-channel leakage of individual tiles. Tile areas of RRAM-based PIM prototype chips vary due to different array sizes

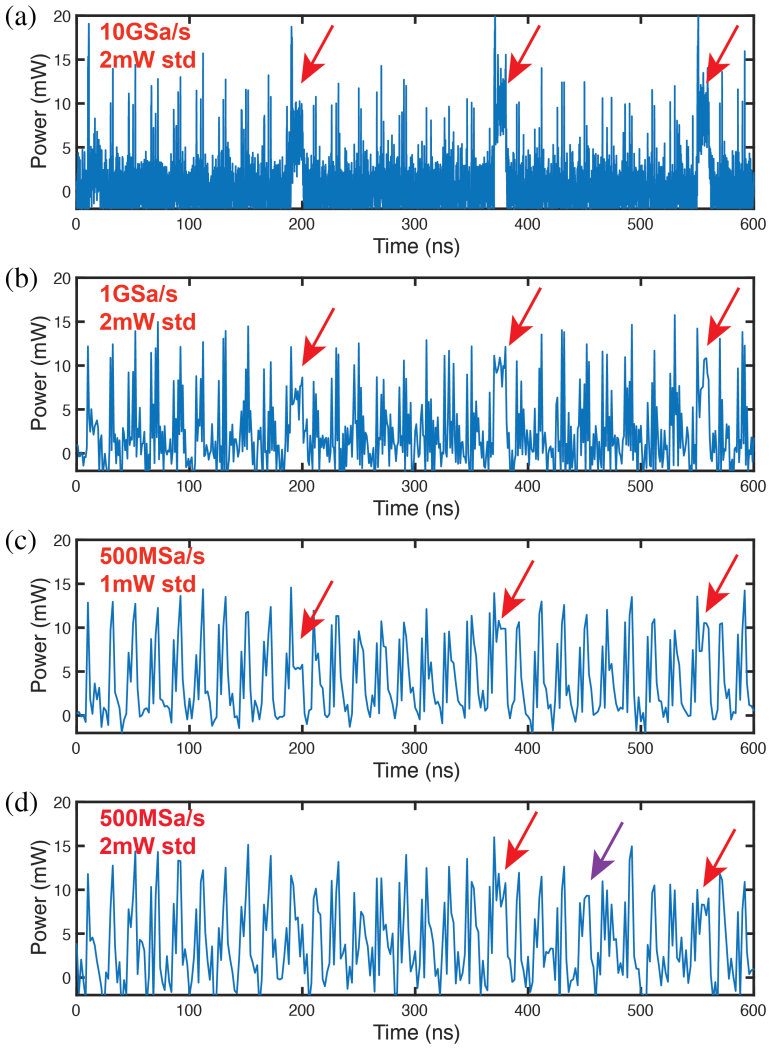


Figure 4.15: Power traces of the same tile for convolution execution with different sampling rates and noise levels. The sampling rates and noise standard deviation levels are labeled in (a) – (d).

and peripheral circuit designs, but most demonstrated tile areas are on the millimeter scale [155] [156] [157] [158]. For example, the reported overall area of a tile with 128×128 array size, same as our system configuration, is $0.5 \times 0.5\ mm^2$ [156]. The spatial resolution of electromagnetic probes can achieve sub-millimeter [159] [160] [161], to make it possible to measure the leakage signal of a tile with no overlapping of adjacent tiles. The second technique is measuring the signal through the power lines of each tile directly. Such techniques have been experimentally used to attack a spin-transfer torque MRAM system for Advanced Encryption Standard (AES) executions [135] [162]. Like MRAM systems, RRAM devices are fabricated in the back-end-of-line so etching

away the passivation layers can potentially provide immediate access to top-level metal lines for probing. Off-the-shelf oscilloscopes can offer 256 GSa/s sampling rate and noise floor of hundreds of microvolts [163]. Such equipment can be used to extract the power data.

We further analyzed the proposed attacking scheme after considering real-world artifacts, sampling rate and electrical noise. Algorithm 1 is based on extracting the total execution time of each tile. The execution can be easily distinguished from the idle state, and the convolution execution is much longer than the fully connected layer execution. Hence, we expect this timing analysis is robust even with artifacts. Algorithm 4 is an analysis of other extracted results without new physical signal measurements, so signal disturbance will not affect it. However, Algorithm 2 and Algorithm 3, which are based on timing and power analysis, may be affected by the artifacts. Low sampling rate and high noise level may make it difficult to distinguish the ADC execution period from the analog computation period of the crossbar array. Timing analysis in Algorithm 2 will fail in this scenario. These conditions will also lead to an unreliable average power estimate, which may affect the power analysis in Algorithm 3.

Table 4.3: Algorithm 2 results with non-ideality.

Noise Std / Sample Rate	0	1 mW	2 mW	3 mW
10 GSa/s	Success	Success	Success	Fail at FC
1 GSa/s	Success	Success	Fail at FC	Fail at FC
500 MSa/s	Success	Fail at FC	Fail	Fail
200 MSa/s	Fail at FC	Fail	Fail	Fail

To test the robustness of the proposed side-channel attack approach in real-word scenarios, we simulated power traces with different sampling rates and noise levels. In our initial simulation power traces without artifact injection are sampled at a rate of 10 GSa/s, and the analog operation and ADC execution periods exhibit significant differences, as shown in Figure 4.10(b) and (c). In

Figure 4.15, the power traces of the same tile for convolution execution with different sampling rates and noise levels are displayed. Thanks to the time required to achieve a stable read output in the analog operation, the analog operation can still be identified even with these artifacts considered, as indicated by the red arrows. The ADC execution time can then be calculated from the intervals of these analog operations to execute Algorithm 2. We noticed that at low sampling rate rates or high noise levels, the ADC and analog operation power signatures can become indistinguishable, as shown in Figure 4.15(d). In Table 4.3, we summarized the failure/success of Algorithm 2 at different artifact levels. We found the attack is more likely to fail at fully connected layers where VMMs are only executed once compared to repeated executions in convolution layers. Hence, there are fewer opportunities to identify the correct analog array execution period at fully connected layers.

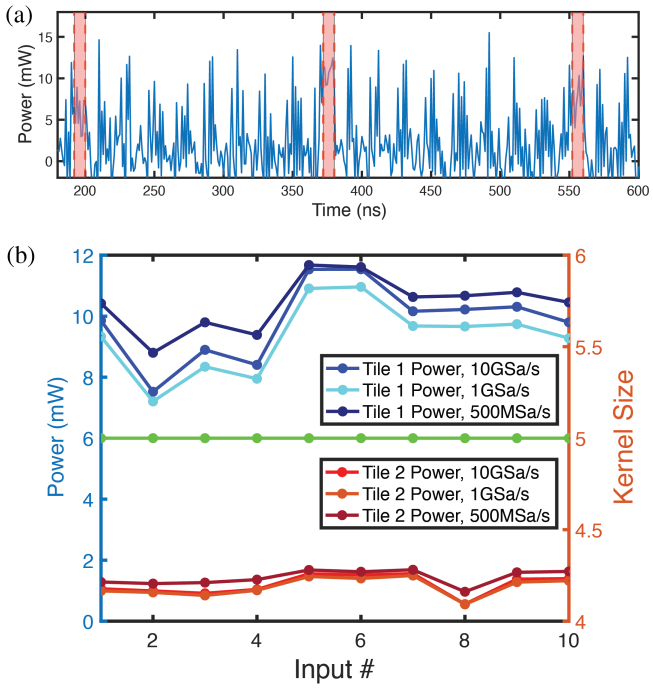


Figure 4.16: (a) Power traces for a convolution tile with 1GSa/s sampling rate and 2 mW noise standard deviation. The red region is the interested period for average power computation. (b) Average power of 2 convolution tiles and predicted kernel sizes with respect to different input images at 2 mW noise standard deviation and sampling rates from 500 MSa/s to 10 GSa/s.

The success of Algorithm 2 is the prerequisite for Algorithm 3, since we need to identify the analog operation region before extracting its average power, as illustrated in the red regions of

Figure 4.16(a). We performed kernel size extraction tasks following Algorithm 3, in the presence of artifacts injection. Even at noise levels of 2 mW standard deviation, the correct kernel size can still be extracted at sampling rates from 500 MSa/s to 10 GSa/s, as shown in Figure 4.16(b). The failure/success of Algorithm 3 at different sampling rates and noise levels are summarized in Table 4.4. An interesting result is when the sampling rate is reduced to 500 MSa/s, where it becomes difficult to identify the analog operation period, as shown in Figure 4.15(d). However, if assuming the average power can still be calculated, the correct kernel size can still be obtained (Figure 4.16 (b)). These results prove the kernel size extraction method is robust against artifacts, likely due to the fact that effect of white noise can be effectively averaged out across multiple convolution executions.

Table 4.4: Algorithm 3 results with non-ideality.

Noise Std / Sample Rate	0	1 mW	2 mW	3 mW
10 GSa/s	Success	Success	Success	Success
1 GSa/s	Success	Success	Success	Success
500 MSa/s	Success	Success	-	-
200 MSa/s	Fail	-	-	-

In summary, the timing and power side-channel attack methods are robust after considering real-world non-idealities, such as sampling rate and electrical noise. Table 3 and Table 4 summarize the required sampling rate for the proposed approaches to work well at different noise levels. The required specs can be offered by off-the-shelf measurement equipment [163]. The required sampling rate can be further relaxed if the attacker can control the clock frequency of the system or effectively filter out noise from the measured signals.

4.6 Countermeasures

Techniques for preventing side-channel attack can be based on eliminating the correlation between the leaked information and the secret information [164], which is the DNN model in our case. For the proposed attacking algorithms, the countermeasures fall into three categories.

The first approach is to eliminate sensitive information leaked from timing analysis. For instance, scrambling the start time of the tiles or invoking dummy tiles with random delay times can obfuscate the layer sequence analysis. If the execution time of tiles are scrambled, we may find all tiles has the same start/end time and execution duration. This will make it impossible to differentiate the layer sequence and layer type. Inserting dummy input bits can increase the Conv layer execution latency. Dummy inputs can make it impractical to retrieve the output feature size from the execution time analysis, and make the iterative algorithm that searches the padding size ineffective. However, the above-mentioned techniques add penalties to both power consumption and latency of the system.

The second approach is to eliminate sensitive information leaked from ADC execution. The ADC execution time is directly related to the number of mapped columns in a tile and used to extract the layer size. Thus, one can add fake inputs to the ADCs in the partially mapped tile to match the ADC execution time as fully mapped tiles. Tiles will all have the same ADC execution time and the attack method based on ADC execution latency will fail. This approach will increase latency and the ADC power consumption.

The final approach is to mask the crossbar power consumption differences among tiles used in the same DNN layer, which will make the kernel size extraction algorithm ineffective. A potential technique is to add dummy conductance values to devices in unused columns. However, this method does not work if all tiles are column-wise fully mapped. Another approach is to devise other mapping schemes to balance and equalize the power consumption at analog computing. For example, thermal-aware weight mapping methods discussed in [142] shows potential in balancing and reducing power consumption. Equalization techniques reduce the power leakage information by creating a consistent power side-channel profile. Equalization has been proved reliable to enhance the resistance of AES engines to power side-channel attacks [165] [166] [167]. One can

apply different weight mapping methods to different tiles to conceal the correlation between power consumption and mapped rows. However, this will introduce more workload in data pre-processing.

4.7 Conclusion

DNN accelerators are increasingly susceptible to malicious model extraction attacks, which expose the network architectural information. Several initial studies on model extraction attacks have been proposed on GPU, CPU and other DNN accelerator platforms [83] [79] [84] [85] [86][168] . DeepSniffer [79] and ReverseCNN [84] extract DNN models from GPUs and general DNN accelerators. However, their attack model heavily relies on data movement in the memory bus, and cannot be generalized over to PIM architecture. Model extraction attacks with power or electromagnetic side-channel attack have also been reported on ARM cortex-based systems and FPGAs [83] [85] [86]. This work first perform model extraction attack on analog PIM-based DNN accelerators.

While mixed-signal PIM architectures may potentially assist with the obfuscation of sensitive data by reducing the degree of data movement and limit memory access, we demonstrate how measurable electrical characteristics can still pose a security vulnerability. To perform reliable power trace analysis, we developed a dynamic power simulator based on a TSMC 28 nm process. We scrutinized the security vulnerability of analog PIM systems for DNN inference acceleration by proposing a series of side-channel attack analysis algorithms. Our analysis showed it is possible to uncover the complete model architecture information without any prior knowledge of the neural network model.

Using the proposed techniques, we were able to systematically uncover all layers in the example model and successfully reconstruct the full neural network model. The proposed approach showed it is feasible to probe a PIM inference engine using algorithms that can work with other convolutional and dense DNN architectures. This study highlights the nature of security patches that may be required at the hardware abstraction, such as scrambling the timing information, adding dummy

cycles to the ADC, and masking crossbar power using different weight mapping methods.

CHAPTER 5

PowerGAN: A Machine Learning Approach for Power Side-Channel Attack on PIM Accelerators

In Chapter 4, we have shown that the DNN architecture mapped on RRAM-based PIM accelerator can be fully extracted from side-channel power and timing leakage. This chapter will focus on data privacy concerns at the user end. As shown in Figure 5.1, a potential security vulnerability is identified in analog PIM systems wherein an adversary can reconstruct the user's private input data from a power side-channel attack. Machine learning-based attack methodology will be shown to achieve high quality data reconstruction from power leakage measurements, even at large noise

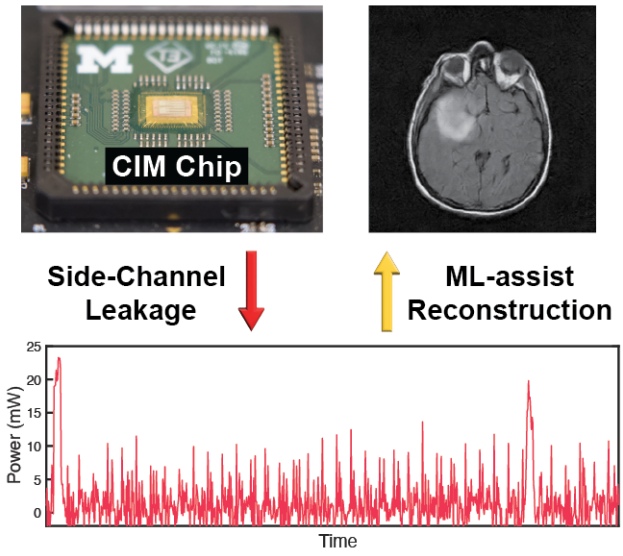


Figure 5.1: Schematic of privacy breach by side-channel profiling.

levels and after countermeasures [34].

5.1 Background and Motivation

5.1.1 Data Privacy in Machine Learning

Machine learning, especially deep neural networks (DNNs), are being used in a broad range of applications, including language processing, computer vision, speech recognition and financial analysis [98][106][169][170]. As the usage of DNN models expands, the importance of data security grows. Attacks on DNN data used in critical applications such as medical diagnosis, autonomous driving, and financial transactions can compromise user privacy as well as proprietary algorithm information [78]. Protecting user data privacy in machine learning is not only an ethical imperative but also a legal requirement and a strategic necessity for building and maintaining trust, ensuring fairness, and promoting the responsible use of technology. The main areas of concern in DNN security include model extraction, adversarial attacks, and privacy breaches [33][78][79][80]. These security attacks on DNNs have been extensively evaluated on systems such as GPUs, CPUs and FPGAs [79] [81] [83] [84] [85] [86], security and vulnerability analysis of analog PIM accelerators is largely lacking.

In Chapter 4, a model extraction attack methodology based on side-channel leakage analysis has been proposed for RRAM-based analog PIM architecture. The users' private input data is also of considerable importance. As a potential solution for prevalent low-power edge-based computing, understanding security vulnerabilities of PIM systems including data privacy becomes paramount. In this chapter, we first propose a machine learning-based approach to reconstruct users' private input data from power side-channel profiling of PIM systems, without prior knowledge of the DNN model mapped on chip.

5.1.2 PIM Architecture for Medical Image Processing

In recent years, RRAM-based PIM systems have been widely studied for DNN inference applications due to their ability to perform in-situ single-step VMM through bit-line current summation [4][76] [74]. One area that has greatly benefited from this technology is computer vision, specifically CNNs, which require intensive VMM operations. Moreover, the RRAM-based PIM systems can incorporate transposed convolution for up-sampling in encoder-decoder networks [171] [172]. This has been shown to improve efficiency and support various DNN models, as evidenced by several studies [173] [174] [175] [176].

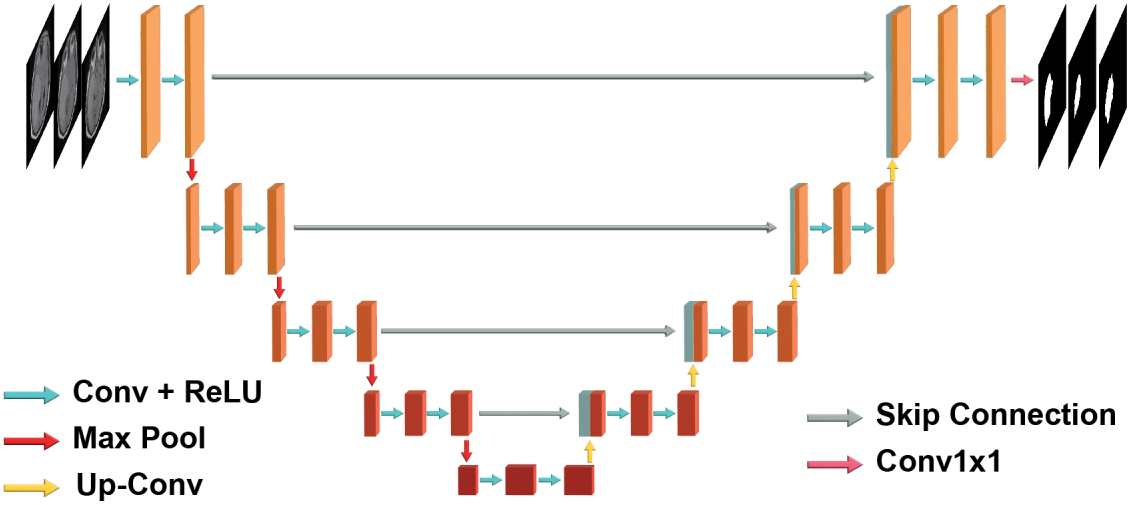


Figure 5.2: Schematic of the medical imaging U-Net model used in this analysis.

Medical diagnosis, which is a subset of computer vision, involves medical image reconstruction, segmentation, and super-resolution [177], and PIM schemes are promising platforms for medical image processing [178]. The U-Net architecture, one of the most widely used CNN architectures, has been extensively used for image segmentation tasks [179] [180]. U-Net, shown in Figure 5.2, has a "U"-shaped architecture composed of a down-sampling encoder and an up-sampling decoder, which perform feature extraction and reconstruction through convolution and transposed convolution layers, respectively. The skip connections in U-Net connect down-sampling and up-sampling paths by concatenating the feature maps to preserve spatial information and improve accuracy. U-Net has been proven effective for multiple medical image segmentation tasks, such

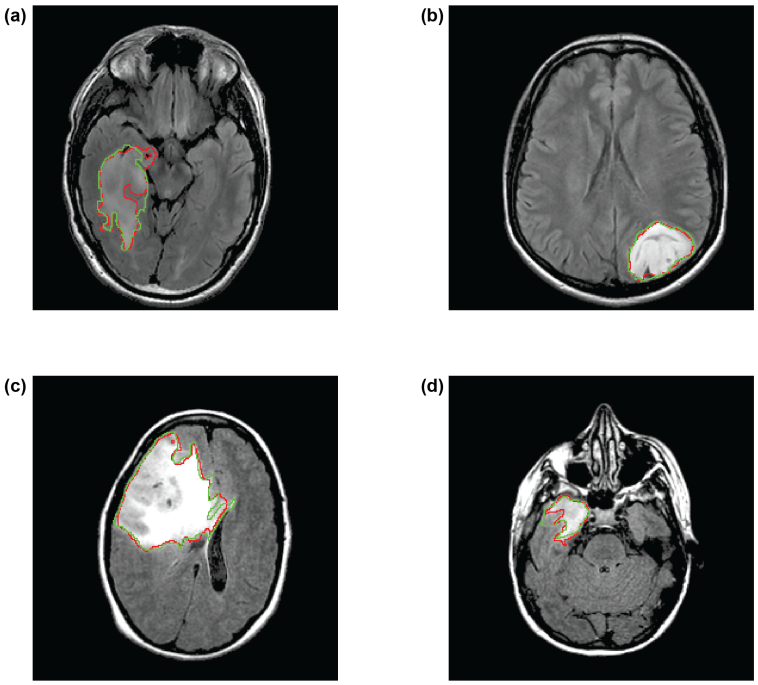


Figure 5.3: U-Net brain MRI image segmentation results.

as identifying tumors or lesions in magnetic resonance imaging (MRI) scans, even with limited training data. Figure 5.3 shows the output results of the U-Net, where the green line represents the ground truth and the red line represents the network prediction results.

In this study, U-Net for MRI segmentation contains the following layers, **U-Net Encoder:** C32–C32–MP–C64–C64–MP–C128–C128–MP–C256–C256–MP–C512; **U-Net Decoder:** C512–C256–C256–C256–C128–C128–C128–C64–C64–C64–C32–C32–C32, where C is for convolution layer, followed numbers are channel depth and MP is for max pooling layer. The MRI dataset used in this study is from [181].

5.1.3 Problem Statement

PIM systems use pre-trained models for inference acceleration on chip. Any potential security threat can be exploited by a malicious adversary who could launch a side-channel attack on the system through the side-channel leakage. Side-channel attacks aim at extracting private data from a hardware system by measuring and analyzing physical parameters during execution [82] [164].

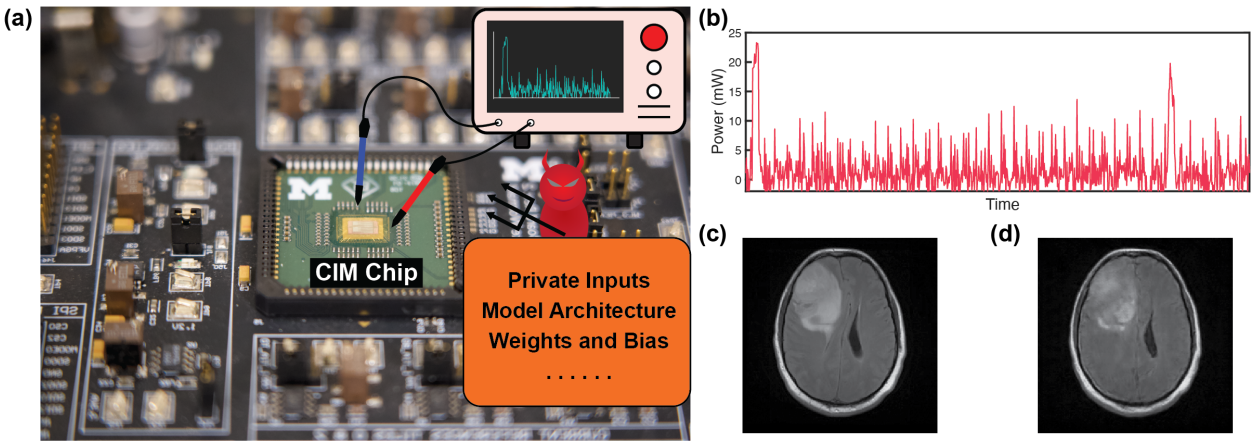


Figure 5.4: (a) PIM chip and the system board. An adversary can potentially perform power side-channel attacks through power trace measurements on the chip's power lines. (b) A representative power trace of a PIM accelerator at inference runtime. Examples of (c) a user's private input sent to the PIM inference accelerator running the U-Net, and (d) reconstructed image from the power side-channel attack using techniques proposed in this section.

Such parameters include supply power, execution time, and electromagnetic emission. Attackers can reverse engineer the sensitive data or architectural information by deliberately measuring and analyzing the side-channel dissipation of the chip. As RRAM devices are fabricated in the back-end-of-line, invasively etching away the passivation layers of PIM macros can provide immediate access to the top-level metal lines of each tile for potential probing. The adversary can then probe the power lines directly and extract the power data using an oscilloscope, as shown in Figure 5.4(a). The side-channel attack is performed on measured leaked traces, with one such example shown in Figure 5.4(b), where the power trace of a PIM module at inference runtime is shown. In this work, the goal is to reconstruct the user's private input data from power traces measurements. Figure 5.4(c) and (d) show an example of the original private input and the reconstruction result using the proposed approach, respectively.

Accurately measuring side-channel leakage signals requires sophisticated data acquisition scheme design and high-precision measurement equipment. To study the vulnerability of the chip design and secure it iteratively, it is more practical to simulate the attack scenarios using real device measurement data, followed by redesigning the chip with security considerations before massive production, especially for the emerging PIM systems. In this chapter, the hardware config-

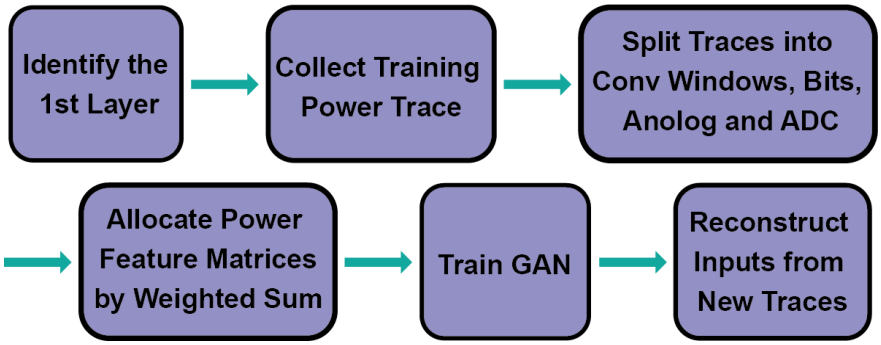


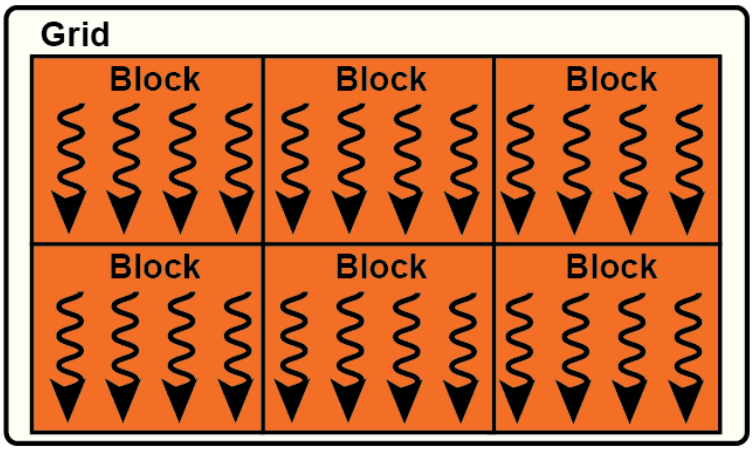
Figure 5.5: Overview of the power side-channel attack flow for private input data reconstruction.

uration, adversarial knowledge assumption and simulation logic are the same as in Chapter 4 and [33]. To fast generate power feature information, we developed a fast power feature simulator using GPU to process every bit input in the bit-serial fashion, which will be discussed later.

5.2 Side-channel Privacy Leakage

5.2.1 Attacking Flow and Power Modeling

Figure 5.5 illustrates the attack flow aimed at reconstructing private input data from the power side-channel leakage. The attack targets the first layer of the DNN model since it is the closest layer to the input port and directly executes the input data. The methodologies proposed in Chapter 4 can be used to extract the property of the first layer, i.e., structure of the convolution layer and the associated PIM tiles used to execute the layer. As DNN models are trained on specific tasks, an adversary can collect power traces using similar input data and learn the dependency between them. In this case, the adversary can feed other MRI images and collect their own power traces to train an attack model before attacking unknown inputs. Once power traces are allocated, power feature extraction and data preprocessing are required to find the correlation between input and leakage. We refer to the data after preprocessing as power feature matrices, which can be utilized to train a machine learning model for input data reconstruction. In our study, we employed a generative adversarial network (GAN) for reconstruction since it shows good noise tolerance and




```
__global__ void powerArray(int* input, float* weight,
  float* power);

__global__ void energyADC(int* input, float* weight,
  float* lut, float* energy);
```

Figure 5.6: Block diagram utilizing CUDA SIMT execution for fast simulation of the crossbar analog computing power and ADC switching energy.

can overcome noise-injection countermeasures.

The power trace simulator proposed in Chapter 4 provides valuable insights into dynamic power and timing data. However, processing every single data point in a large dataset is impractical due to the time-consuming nature of simulating power dissipation at sub-nanosecond-level precision, as well as the generation of tremendous data files. To address this, we developed a fast power feature simulator based on NVIDIA Compute Unified Device Architecture (CUDA), leveraging modern GPU's single instruction multiple threads (SIMT) to process every bit input in the bit-serial fashion, as shown in Figure 5.6. The CUDA kernel functions simulate power dissipation and total energy of RRAM arrays and ADCs at each execution, respectively. Both kernels take bit-serial inputs and conductance weight matrices as inputs. The kernel function for array power simulation computes the power using applied input voltage and device conductance values. The kernel function for ADC is more complex. First, we compute the switching energy of each output code (Figure 4.4(b)) and store them in a lookup table (LUT). Then, the kernel function computes the analog current of each bit-line, followed by scaling it into the proper range to index the ADC energy from the LUT. Due

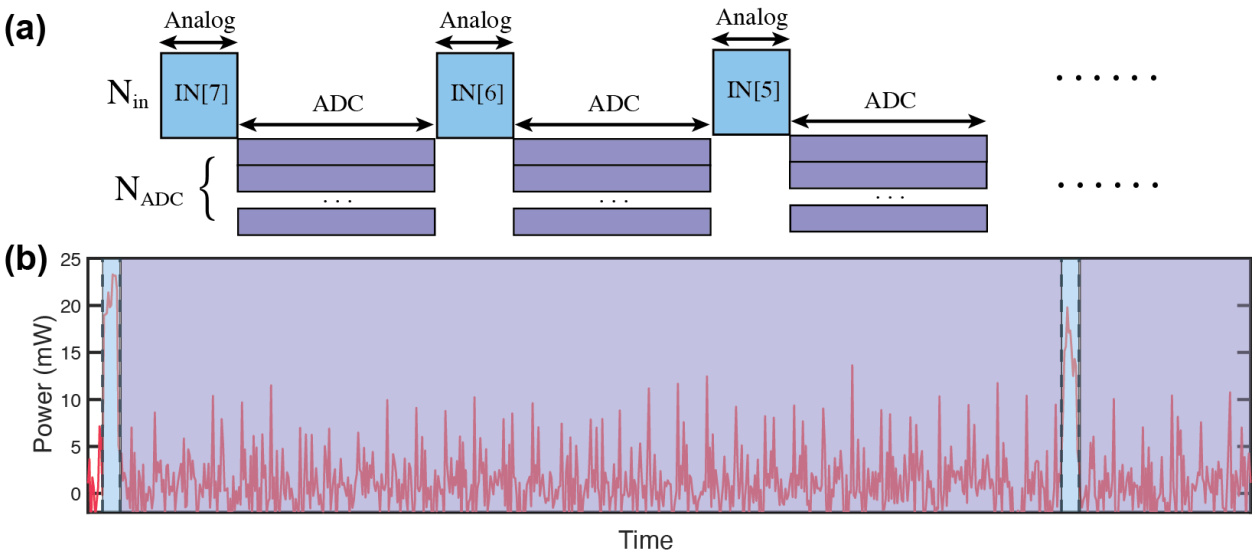


Figure 5.7: (a) Timing diagram of a PIM tile at inference runtime. (b) The simulated power traces of a PIM tile during execution.

to ADC sharing, the ADC energy in one execution is the sum of four ADCs.

To convert all outputs in an array with 128 columns and 4 ADCs requires 32 executions. However, the ADC traces are similar to each other, making it impractical to distinguish every execution with measurement noise, as shown in the noisy trace Figure 5.7(b). Therefore, we treat the energy of all 32 ADC executions as a whole for further data preprocessing.

5.2.2 Privacy Leakage

When an input image is processed by the model stored on the PIM chip, the convolution window in the first convolution layer slides through the entire input image. Here we define two power feature matrices, corresponding to the analog array computation and ADC conversion energy when performing the convolution operation, respectively. The power feature matrices will have the same size as the output feature map of the first convolution layer. An entry in a power feature matrix corresponds to the collected power information during the convolution operation at the corresponding position in the input feature map. Within each convolution window, the input data are reshaped and scaled into 8 bits before being applied to the crossbar array. Processing an input bit

in PIM can be divided in two steps: analog array computation and analog-to-digital conversion, as shown in Figure 5.7(a). Because the eight input bits are of varying significancy, the input-dependent power leakage at each bit computation should not be treated equally. To recover the input-dependent power data, we used a weighted sum approach for the array power and ADC energy according to the bit significance, as shown in Equation 5.1, 5.2 and Figure 5.8.

$$P_{array} = \sum_{i=0}^{7} P_{array}[i] \times 2^i \tag{5.1}$$

$$E_{ADC} = \sum_{i=0}^{7} E_{ADC}[i] \times 2^i \tag{5.2}$$

Once the weighted sum results of each convolution window have been computed, they are then populated into the two power feature matrices. Figure 5.9(a) and (b) show examples of the obtained array power feature matrix and the ADC power feature matrix, respectively, without considering noise in the power data measurement. The power side-channel leakage exhibits a strong dependency on the input data and reveals a security risk.

Both the array power dissipation and ADC energy consumption are functions of the inputs and weights, which explains the strong dependency between the power side-channel leakage and input data. During inference, the weights are held constant leading to a linear relationship between the power feature matrices and the original input, as shown in Equation 5.3,

$$y_{i,j} = \mathcal{F}\left(\begin{bmatrix} x_{i-r,\ j-r} & \cdots & x_{i-r,\ j+r} \\ \vdots & \ddots & \vdots \\ x_{i+r,\ j-r} & \cdots & x_{i+r,\ j+r} \end{bmatrix}\right) \tag{5.3}$$

where i, j are indices of the entry from power feature matrices, r is the radius of the convolution window, x is the the input data and y is the corresponding entry in the power feature matrices. The function $\mathcal{F}$ is used to convert the input data into power feature data after weighted sum. Regardless of whether $\mathcal{F}$ represents array or ADC, it is always a one-to-one projection. As a result, the input

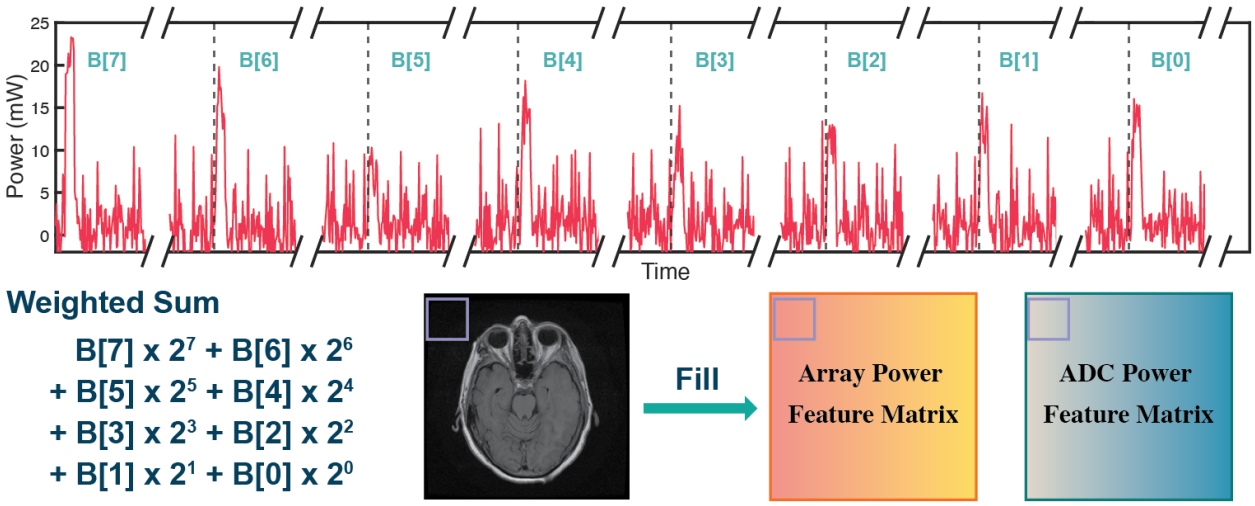


Figure 5.8: Data preprocessing steps after power trace acquisition

information is preserved in the power feature matrices and leading to a severe security issue.

Additionally, as shown in Figure 5.9(a) and (b), the array power feature matrix contains more detailed information than the ADC power feature matrix. This is because each entry in the ADC power feature matrix represents the total energy of 32 executions, with each execution involving four ADCs operating together. Compared with the array computation power feature matrix which entries indicating single execution, the ADC power feature matrix produces coarser granularity. Furthermore, from the weight mapping scheme in Figure 4.5(a), the energy consumption of the four ADCs are associated with a weight value in four representations of MSB+, MSB–, LSB+ and LSB–. Since the LSBs and MSBs have different impacts on the weight value, the ADC energy output is not a direct linear transformation of the input data, where the array power feature matrix corresponds to a linear transformation of the input data (directly proportional to the product of the input and the weight matrix).

5.2.3 Countermeasures

The power side-channel attack approach mentioned above is capable of reverse engineering private inputs. However, for a real-world applications non-ideal effects such as noise during data acquisition must be considered. Noise can originate from multiple sources of the chip, including

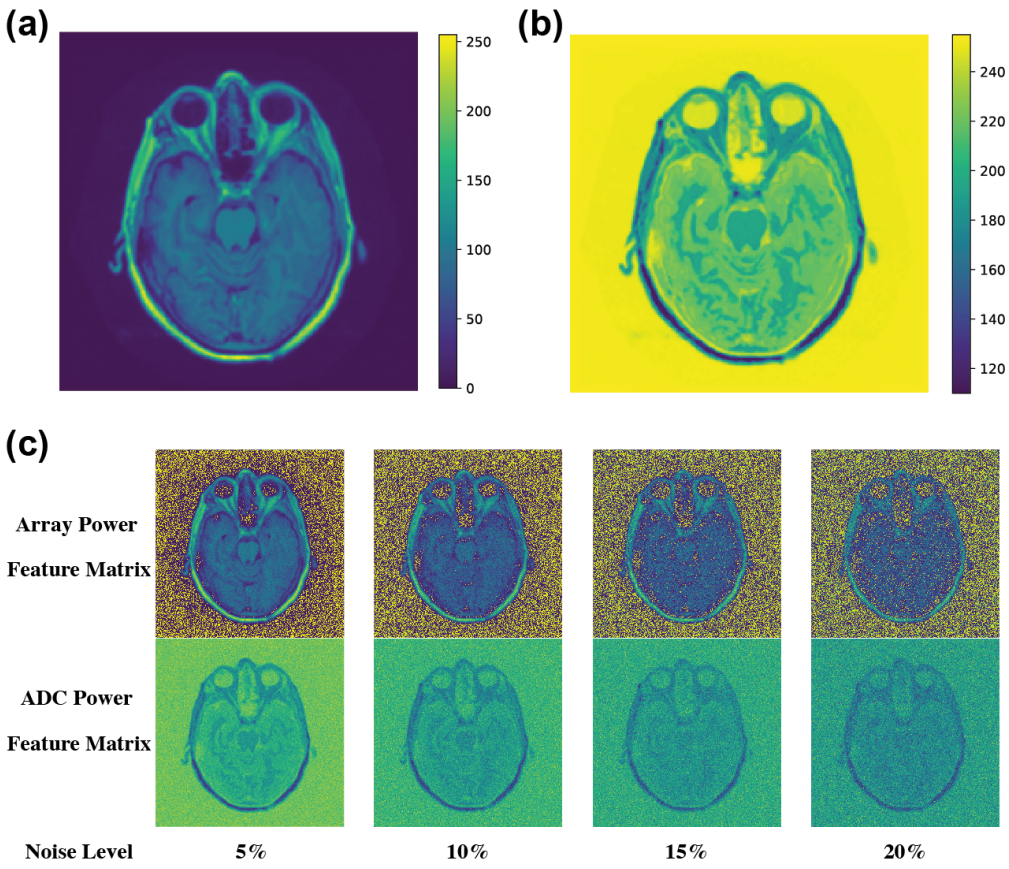


Figure 5.9: (a) Array power feature matrix and (b) ADC power feature matrix obtained in an ideal simulation without adding noise, normalized into 8-bit unsigned integer range. (c) Power feature matrices with different levels of noise injection. The noise level refers to the ratio of the standard deviation to the maximum value in the power feature matrix.

thermal noise and human-made noise [182] [183] [184]. Thermal noise is a physical phenomenon that cannot be eliminated. Thermal noise can be found on RRAM devices. Although ideal capacitors have no thermal noise, when they are coupled with other components in the circuit, there will be a combination of kTC noise. Other noise from power lines or measurement equipment may have higher noise power and will lower the signal-to-noise ratio when the adversary measures the power side-channel leakage. The power feature data, accounting for the presence of noise, are obscured by a noise term N, which can be mathematically described by Equation 5.4.

$$y_{i,j} = \mathcal{F}\left(\begin{bmatrix} x_{i-r,\ j-r} & \cdots & x_{i-r,\ j+r} \\ \vdots & \ddots & \vdots \\ x_{i+r,\ j-r} & \cdots & x_{i+r,\ j+r} \end{bmatrix}\right) + \mathcal{N}(\cdots) \tag{5.4}$$

Noise can also be used to protect the system. One common countermeasure to mitigate power analysis side-channel attacks is noise injection. Noise injection works by adding a random noise signal to the original signal to mask the correlation between the leaked information and the secret information [185] [186]. The noise signal is designed to be random and uncorrelated with the original signals, so statistical analysis methods for denoising may not be valid anymore.

In Figure 5.9(c), we simulate different noise levels from 5% to 20% at side-channel data acquisition. The percentage levels here are defined as the ratio of the noise's standard deviation to the maximum measured signal in the power feature. As the noise level increases, details of the cerebrum region in the power feature matrices are lost. It is noteworthy that when the noise level reaches 20%, the cerebrum region was effectively masked from the power feature matrices, and only noise is circled by the skull. Hence, noise injection is a powerful countermeasure to mitigate the side-channel leakage in the PIM system. To deal with noise injection, at the adversary end, a more effective attack approach is required.

5.3 Machine Learning-Assisted Side-channel Attack

5.3.1 Generative Adversarial Network (GAN)

Adversaries often attempt to design elaborate denoising schemes to stripe the noise signal and expose the valuable original signal [81]. However, using conventional denoising techniques requires considerable effort in denoising design at both the hardware and algorithm levels. The adversary needs to specify the noise frequency and apply a low-pass filter to cut off high-frequency noise during measurement. Then, the adversary needs to identify the working spectrum of the PIM system, and recover the distortion induced by the power measurement circuit. To establish the

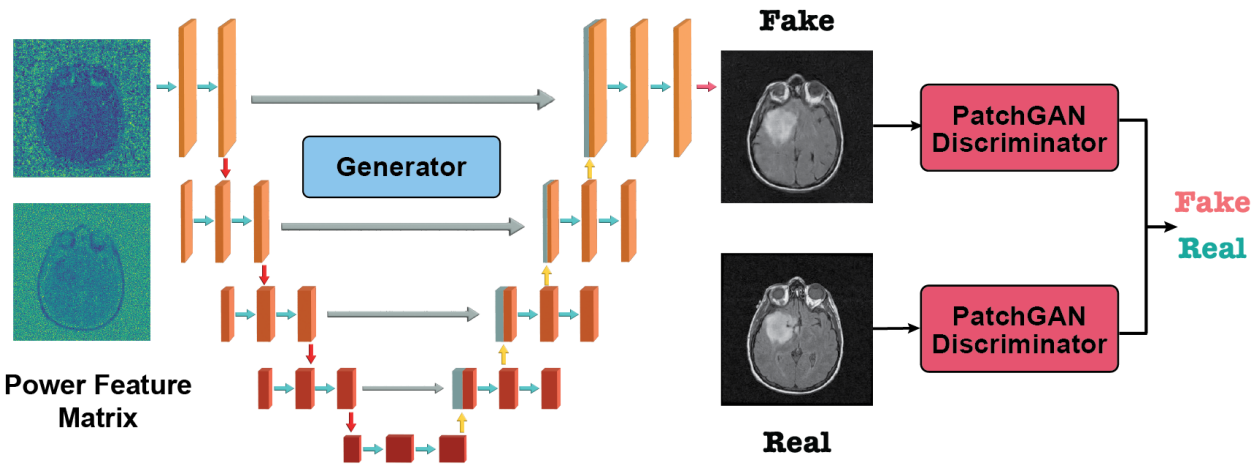


Figure 5.10: Schematic of the pix2pix cGAN for reconstructing image from noisy power traces. The two power feature matrices are concatenated as the input to the generator, which is based on a U-Net architecture. The generated image, along with the original image and the power feature matrices, is fed to the PatchGAN discriminator for differentiation.

relationship between the restored power curve and the original data, the adversary is required to conduct circuit analysis of the power data at each execution frame, which can be inefficient when dealing with large volumes of input. Therefore, a flexible attacking approach with noise tolerance is essential for efficient side-channel attacks and for prompting to designing more secure and reliable PIM systems.

Machine learning approaches have recently been explored for side-channel attacks since they can be highly automated and scalable, allowing attackers to extract sensitive information with minimal human intervention from large volumes of data [187] [188] [189]. Machine learning-based side-channel attacks involve a training phase and an attack phase. The training phase can be controlled by the adversary by building a leakage model from the trace collected earlier using known input data, which alleviates the requirement of collecting sufficient traces from limited resources. The adversary can design a system-specific model to prompt a side-channel attack and recover the target data from newly measured trace during the attack phase.

GANs are a type of DNN architecture consisting of two neural networks for generating synthetic data: a generator and a discriminator [190]. In conventional GANs, the generator takes random noise as input and tries to generate data to mimic the training data, while the discriminator takes

both the real data and generated data and tries to distinguish between them. During training, the two networks are trained together in a min-max game, where the generator tries to produce data that can fool the discriminator, and the discriminator tries to evaluate the authenticity of the generated data. Unlike conventional GANs, which take random noise as input, conditional GANs (cGANs) take extra information as input, such as image features or text description [191] [192] [193] [194], as shown in Figure 5.10. The target of cGANs is to generate more controlled outputs belonging to a certain category or containing certain desired features. cGANs have many potential applications in various fields such as image-to-image translation, text-to-image synthesis, and style transfer. In a side-channel attack, the adversary's goal is to train a neural network to reconstruct the input information from side-channel leakage, making cGANs a good fit for this task.

Noise is required during training of both conventional GANs and cGANs because it provides the generator with a source of randomness, allowing it to produce diverse and realistic data and improve the model performance. Consequently, GANs are excellent candidates for coping with noise (and leveraging the noise) during side-channel data measurement, enabling the elimination of complex conventional denoising schemes for data acquisition. GAN-based DNNs can make side-channel attacks more flexible and enhance attack success rate.

5.3.2 Experiment Setup

We note the reconstruction of the original image from power feature matrices is analogous to image-to-image translation. In this study, we adopted the pip2pix cGAN [193] architecture to reconstruct the input images from side-channel leakage. Pix2pix is a specialized version of cGANs designed to map an input image from one domain to an output image in another domain.

The pix2pix architecture used in our experiment is shown in Figure 5.10. The generator is a standard U-Net architecture, similar to the one discussed in Figure 5.2 for medical image segmentation. The array power feature matrix and ADC power feature matrix, shown above and below in the left of Figure 5.10, are concatenated in the channel direction before being fed into the neural network. The discriminator is a patch-based CNN called PatchGAN. Two discriminator

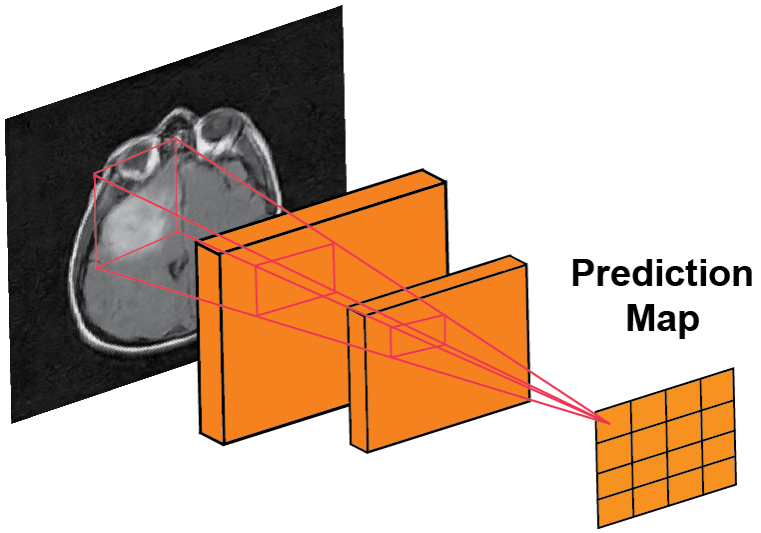


Figure 5.11: The PatchGAN discriminator architecture.

networks are used in the discrimination phase, by analyzing pairs of images, a real MRI image and a fake image generated by the generator, along with the stacked power feature matrices as input, and outputs a patch-level prediction. The architecture of the PatchGAN discriminator is shown in Figure 5.11. For a input image, the PatchGAN takes a patch of the image as input and outputs a matrix of values, where each value in the matrix indicates the probability that the corresponding patch in the input image is real. For the 256×256 images, patch size of 70×70 can achieve realistic reconstruction results. In this case, the prediction map size is 4×4. By computing the average of the values in the prediction map, the PatchGAN can measure the overall realism of the generated images.

PatchGAN has several advantages in reconstructing the lost details in the power feature matrices. Firstly, it encourages the generator to produce more detailed and high-frequency information in the output, as it has to fool the discriminator at the smaller patch level (vs the larger image level). Secondly, the discriminator can provide more fine-grained feedback to the generator since it evaluates images at the patch level. Thirdly, by using smaller patches rather than the entire image, PatchGAN can capture local image features.

Both generator and discriminator use modules of the convolution-BatchNorm-ReLU. The generator takes a U-Net architecture with convolution kernel size of 4×4 with a stride of 2. In the

encoder part of the U-Net, the convolution layer down samples the input by a factor of 2. And in the decoder part, it up samples by a factor of 2 using transpose convolution. The U-Net in the generator consists following layers, where the skip connections concatenate features from layer i to layer $n–i$, which doubles the channel in the decoder and D indicates the layer with a dropout rate of 50%. The generator U-Net contains the following layers, **Generator Encoder:** C64–C128–C256–C512–C512–C512–C512–C512, **Generator Decoder:** CD512–CD1024–CD1024–CD1024–CD1024–C512–C256–C128, where CD denotes a Convolution-BatchNorm-Dropout-ReLU layer with a dropout rate of 50%.

The power feature matrices and MRI images (generated or orginal) are concatenate before feeding into the discriminator with the architecture of C64-C128-C256-C512, followed by a convolution layer to map to a 1-dimensional output with Sigmoid function.

The brain MRI image dataset is randomly split into train, validation and test with the ratio of 0.8:0.1:0.1. A total of 3143 MRI image data are used in 200 training epochs with batch size of 1. Both original MRI images and power feature matrices are used to train the pix2pix GAN. To incorporate the channel depth of 3 in RGB images, the power feature matrices are concatenate as [Array, ADC, ADC]. The model is trained on an NVIDIA Tesla A40 GPU with 200 epochs, batch size 1. We applied an Adam solver with a learning rate of 0.0002 and momentum parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$.

5.3.3 Result and Discussion

Figure 5.12 shows the image reconstruction results as a function of training epochs. The test image used for reconstruction was injected with a 20% noise level in the power feature matrices. The model is capable of generating a high-level brain structure, including a highlighted tumor region, in just 5 epochs. However, some tiling artifacts can be observed in the reconstructed images when the epoch number is below 20, and the detailed information of brain lobes has not been fully reconstructed. The model is fine-tuned in the subsequent epochs to achieve more precise reconstructions of the MRI images. Human brains are characterized by a folded cerebral cortex,

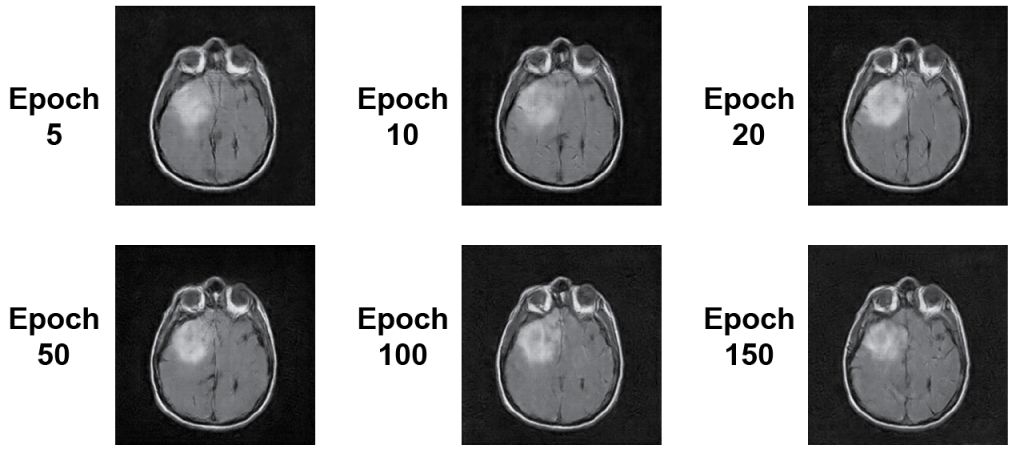


Figure 5.12: An example showing evolution of the reconstructed images from power feature matrices with a high noise level of 20%, over 150 training epochs.

which exhibits diverse details across different MRI images, which makes it hard to reconstruct all detailed information. As the epoch number increases, the model was able to reconstruct more details of sulci and gyri in the cerebral cortex, along with more precise locations of the tumor region.

Figure 5.13 shows reconstruction results for brain MRI images corresponding to different horizontal sections of the brain scan, and different tumor types and brain lobes structures, with varying levels of noise during measurements. The original images are shown on the left, and the reconstructed counterparts with different noise levels are shown on the right. All the reconstructed images demonstrate accurately restored large-scale structures, including tumor type, tumor region, and brain lobes structures. Furthermore, local details such as sulci and gyri in the cerebral cortex are also well preserved in the reconstruction results, and closely resemble the original MRI images. The quality of the reconstruction is evaluated using the structure similarity (SSIM) value [195], which is calculated using a Python package `skimage.metrics.structure_similarity()`. SSIM replicates the behavior of human visual perception system, which is highly adapted for extracting structural information from a scene. SSIM evaluates images based on three comparison measures: luminance, contrast and structure. SSIM value ranges from –1 to 1, where 1 indicates perfect similarity, 0 indicates no similarity and –1 indicates perfect anti-correlation. SSIM values of the

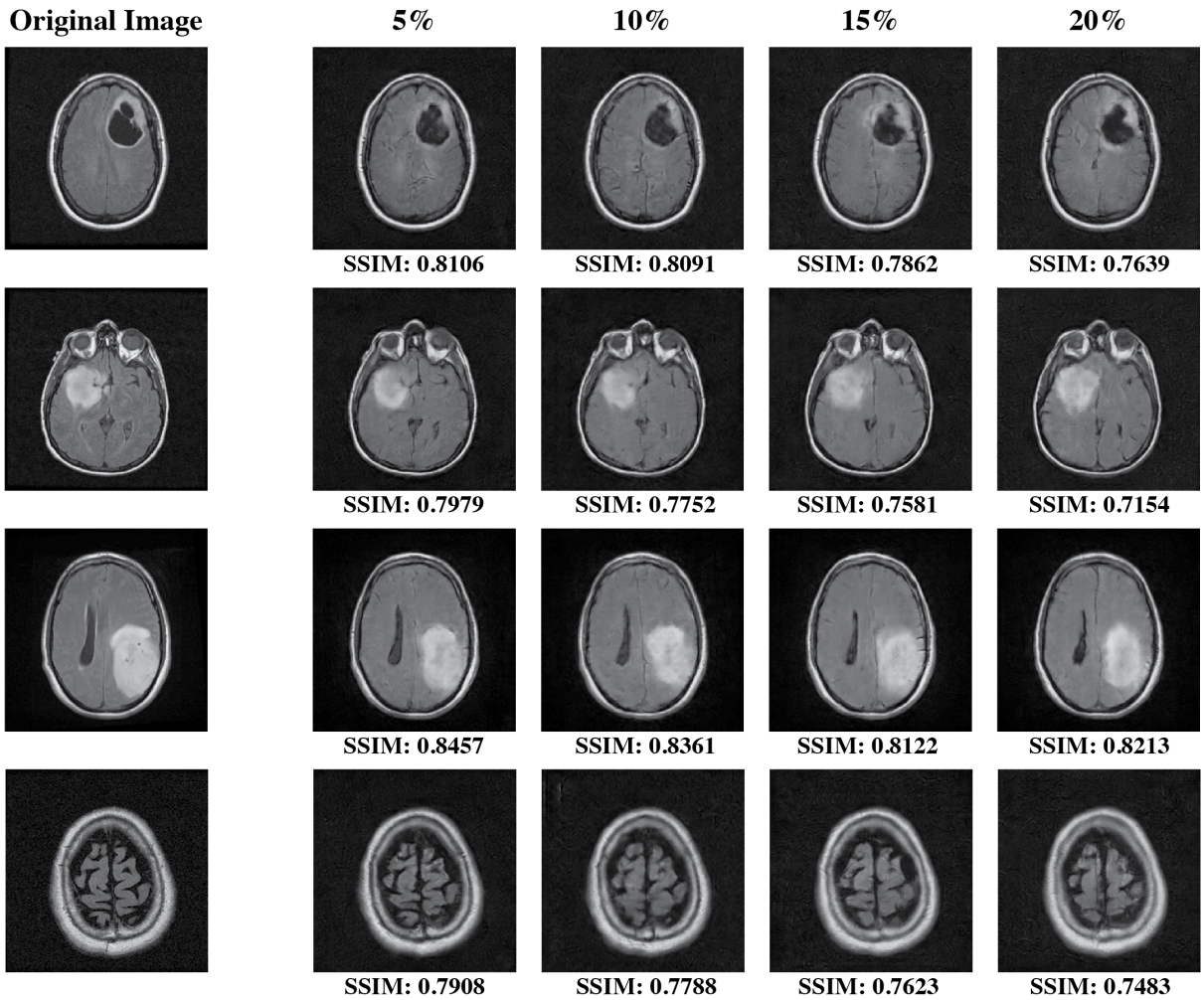


Figure 5.13: Image reconstruction of four representative brain MRI images. The left column shows the original MRI images, and the right columns show reconstruction results from power traces measured at different injected noise levels.

example images are shown in Figure 5.13. We calculate SSIM values of all 393 test cases with different noise levels during power trace acquisition. The results are summarized in Table 5.1. To validate the efficacy of the input reconstruction approach, we compare the reconstruction SSIM values with SSIM values from noisy images by adding Gaussian noise with the same standard deviation to the original images directly. The reconstruction SSIM values show significantly better results over the noise images, as shown in Table 5.1. It should be noted that as the noise level increases, the quality of the reconstructed images slighted decreases, leading to blurred images and missing or added sulci and gyri. However, most of significant information from the original

images remains intact, even at high noise levels of up to 20%. Consequently, the cGAN-assisted side-channel attack is effective in defeating this level of noise injection countermeasures in PIM systems.

Table 5.1: SSIM of reconstruction results and image with Gaussian noise.

Noise Level	5%	10%	15%	20%
SSIM (reconstruction)	0.8193	0.7993	0.7826	0.7505
SSIM (noise image)	0.4295	0.2079	0.1286	0.0889

5.4 Conclusion

In this chapter, we analyzed power side-channel attacks on PIM accelerators, and show carefully designed side-channel attacks can reverse engineer private input data from the user without any prior knowledge of the DNN model used on the chip, thus revealing a potential significant security vulnerability. We propose an automated input reconstruction scheme based on pix2pix cGAN for input image reconstruction from power side-channel leakage, and demonstrate the effectiveness of the proposed attack method on a brain MRI dataset. Our experiments show the power side-channel attack can tolerate a high noise level in power data acquisition, and defeat conventional noise-injection countermeasures. This work highlights a critical vulnerability in PIM systems and underscores the need for greater attention to security considerations in PIM architecture design.

CHAPTER 6

Physical Unclonable Function Systems Based on Pattern Transfer of Fingerprint-like Patterns

In Chapter 4 and 5, we have discussed PIM architecture security vulnerability and potential countermeasures. To secure chips from intrinsic physical entity, physical unclonable functions (PUFs) are potential candidates. A PUF exploits inherent random variations introduced by manufacturing processes to form secret keys on the fly [196]. In this chapter, we demonstrate a PUF system based on fingerprint-like random planar structures through pattern transfer of self-assembled binary polymer mixtures [197]. The proposed fingerprint PUF is compatible with back-end-of-line (BEOL) process and offers potential for hardware security primitive in IoT industry.

6.1 Background and Motivation

Hardware secure systems used for identification, authentication, private key generation, anti-counterfeiting as well as advanced protocols are highly desired [198][199]. PUFs are important hardware security primitives that have been increasingly used as the hardware root-of-trust for securing chips [196][200]. A PUF exploits the variations introduced during the manufacturing process as an entropy source to offer a unique "fingerprint" to each individual chip. When applying a challenge (input) to a PUF, it will generate a unique and unpredictable response (output). As challenge-response pairs (CRPs) are device specific, the PUF technique can be used for authentication and private key generation, as shown in Figure 6.1. So far, numerous PUF

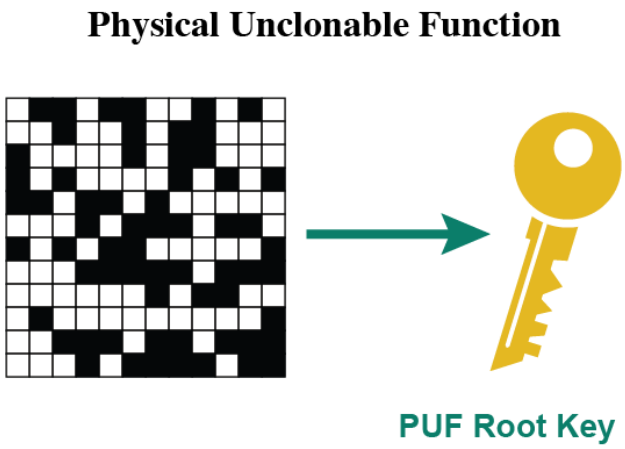


Figure 6.1: PUF for secret key generation.

systems have been constructed, such as optical PUF [201], coating PUF [202], arbiter PUF [203], ring-oscillator PUF [199], SRAM PUF [204][205] as well as non-volatile memory based PUFs [206][207][208][209][210]. However, most of conventional PUFs rely on small variations from unknown factors during fabrication and require complex test schemes, pre-processing or post-processing units to compensate for the minor variation [211][212]. Additionally, PUFs based on active devices suffer from instability as they are sensitive to voltage, temperature, humidity and prone to output errors [213]. Hence, a PUF system based on native properties of passive devices that can be easily implemented and tested is valuable to the semiconductor industry [214].

Figure 6.2 shows the schematic of a secure-aware chip featuring a combination of a PIM engine and a PUF. This innovative design facilitates lightweight and energy-efficient security protocols tailored for edge AI/ML applications. The PUF, with capabilities in authentication and cryptography key generation, ensuring secure access and data protection for PIM engine, guarding against side-channel attacks discussed in Chapter 4 and 5. The control flow for the secure-aware AI chip is shown in Figure 6.2. When PUF serves as an authentication control for the PIM engine, only authorized users can access the system. The secure verification flow initiates by sending a challenge and an external ID to the PUF. Subsequently, the PUF compares the external ID with the internal ID derived from the PUF response. The PIM engine is activated only when the comparison passes, preventing malicious users without a valid ID from waking up the PIM engine.

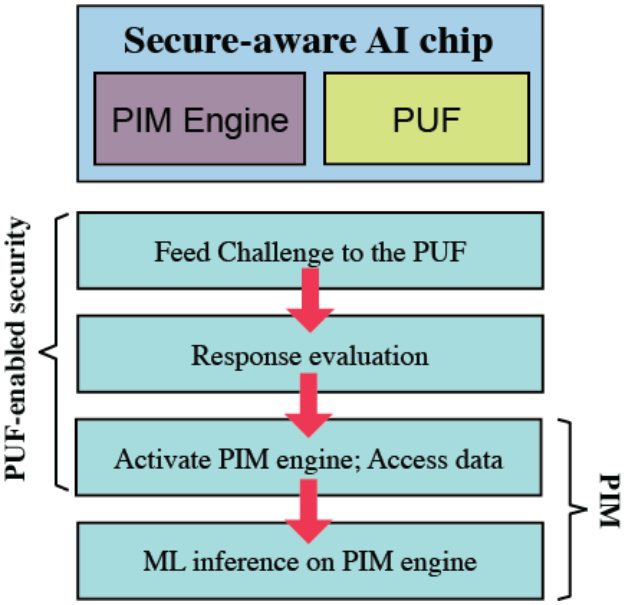


Figure 6.2: Secure PIM chip with the PUF implementation.

This prohibits attackers from querying the system to extract confidential model information, such as model architectures and weights. Certain side-channel leakages are known to be correlated with plaintext. The PUF-generated key can be used to encrypt and decrypt sensitive information, ensuring that only the ciphertext is susceptible to extraction via side-channel attacks. While PEs must execute computations on plaintext, encryption can be applied to the data communication process of PIM engine, thereby mitigating the side-channel leakage at the NoC, I/O buffers and off-chip communications. The ciphertext remains encrypted until the downstream computation commences, offering protection for sensitive data and thwarting attempts by attackers to breach private data.

In this chapter, we develop a PUF system by taking advantage of the unique fingerprint-like features originated from the phase separation of a polystyrene (PS) and poly (methyl methacrylate) (PMMA) polymer mixture during self-assembly. Phase separation of binary polymer mixtures is driven by the system to reduce its total free energy, and results in fingerprint-like or dot-like microstructure [215][216]. The large shape variation is transferred to the conductance variation through pattern transfer. The proposed PUF attains the requirement for uniqueness, entropy, and reliability at high temperature. By tuning the area ratio of the electrode size to the feature size,

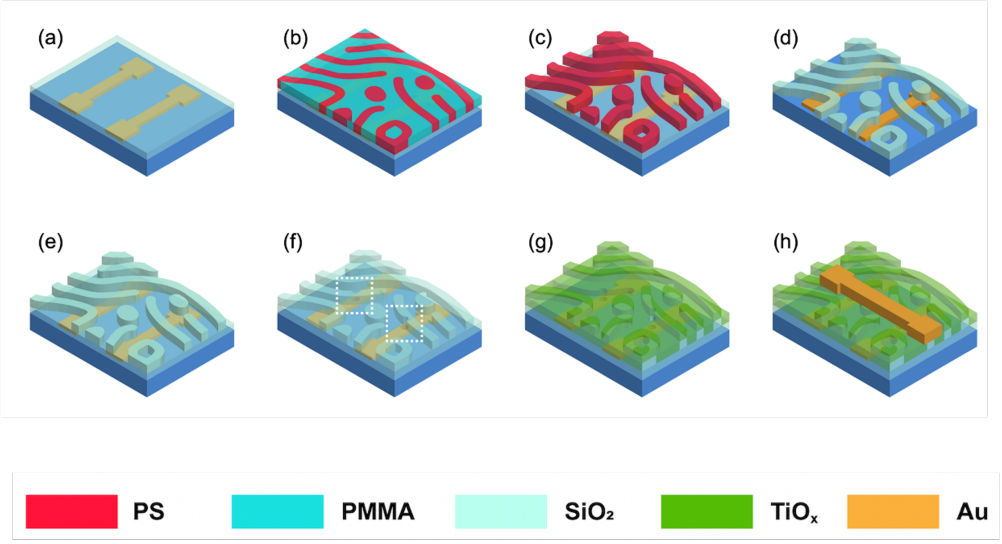


Figure 6.3: PUF device fabrication process.

the proposed fingerprint PUF can function in either differential mode or on/off mode. Since the fingerprint PUF can be integrated at BEOL, it allows additional security features to be added at a separate fab after the front-end processes, providing flexibility and full control for the end users.

6.2 Device Fabrication and Characteristics

The fabrication process is shown in Figure 6.3. The PUF devices are constructed by sandwiching fingerprint patterns between top electrodes (TE) and bottom electrodes (BE). After BE fabrication and SiO_2 film deposition, phase separation of the PS and PMMA domains are formed. The random domains are then transferred to conductive and insulating regions by the following steps. First, PMMA is removed by acetic acid wet etch. Then, the remaining PS acts as a mask for CF_4/CF_3 plasma etch of SiO_2 to transfer the domain patterns. The PS is then removed by oxygen plasma strip. To define the active area in the PUF devices, a blank SiO_2 layer is deposited, followed by a via structure formation through CF_4/CF_3 plasma etch. The active region is defined by the via structure (highlighted in Figure 6.4(a) as the dashed circle). Conductive TiO_x ($x < 2$) is then deposited in the exposed regions in the via (Figure 6.4(a)). Since the conductance of TiO_x is much higher than SiO_2, the fingerprint-like PMMA domain pattern is transferred to the effective conductive area,

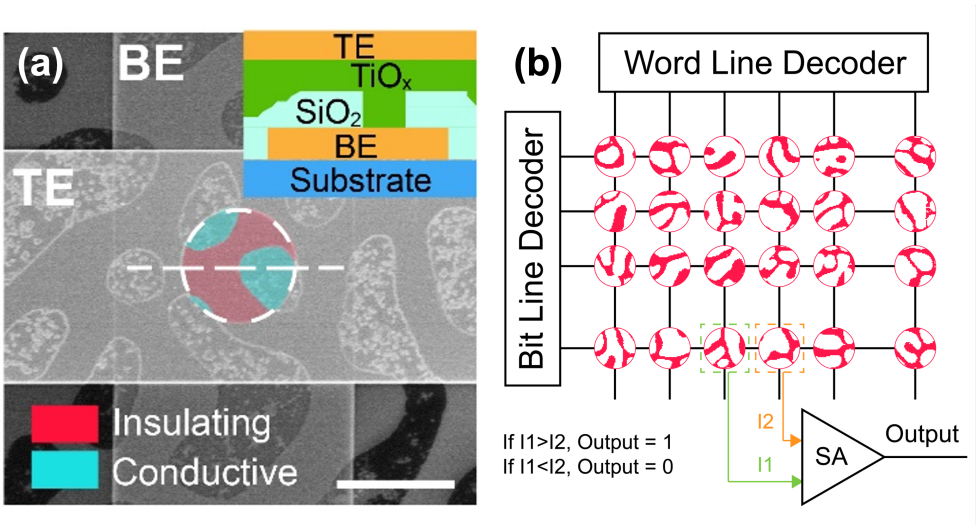


Figure 6.4: (a) Scanning electron micrograph (SEM) image of a device after fabrication. Scale bar: 10 μm. Inset: device cross-section schematic. (b) Schematic of the PUF system. Showing the device pair used to generate a single binary output.

and variations in the PMMA domains are reflected in variations in device conductance values. An SEM image of a device with 10 μm contact is shown in Figure 6.4(a). As the PMMA domain formation is random, the effective conductive TiO_x area of each device is expected to have a large variation. The PUF chip generates random bits using adjacent differential pairs, as Figure 6.4(b) illustrates. The large device conductance variations, originated from the fingerprint pattern, allows us to implement PUF using differential pairs, schematically described in Figure 6.4(b). Briefly, the output current value of two adjacent devices will be read and generate a single bit (1 or 0), depending on the sign of the output current difference. The unique output patterns from CRPs where the challenge is the row addresses, and the response is the binary outputs.

Figure 6.5(a) shows the I–V characteristic of 50 devices from a single chip. The read current at 0.1 V exhibits a large variation range from 20 μA to 100 μA . The conductance values of these devices correlate well with the areas of the TiO_x region, extracted from the SEM images. The SEM images of 4 typical device (labeled in Figure 6.5(b)) are shown in Figure 6.5(c), with the conductive and insulating regions colored blue and red, respectively. Figure 6.5(d) shows the conductance distribution of 350 devices with 10 μm via diameter. Here the diameter of the contact via is about twice of the fingerprint pattern width. The measured conductance exhibits a Gaussian distribution

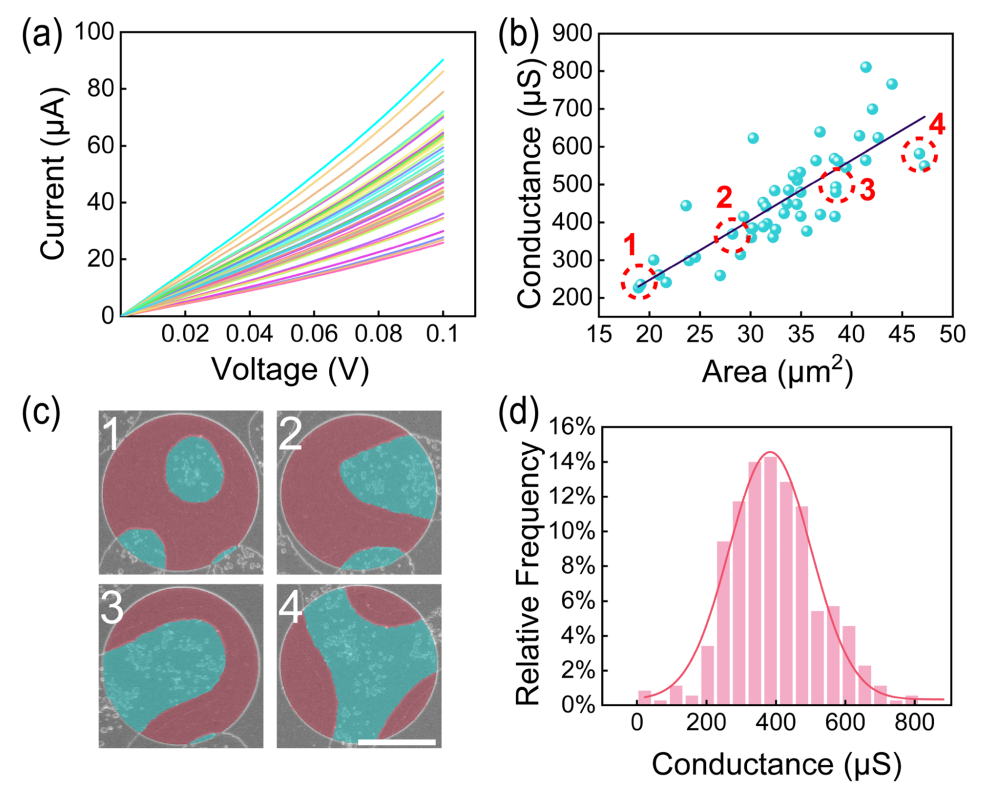


Figure 6.5: (a) I–V characteristic of 50 devices. (b) Conductance values of devices with respect to the exposed TiO_x areas. (c) SEM image of 4 devices in (b) with different exposed TiO_x areas. Scale bar: 5 μm. (d) Histogram and fitting curve of conductance extracted from 350 devices.

with the mean value of 397 μS and standard deviation of 132 μS.

6.3 PUF Construction and Evaluation

6.3.1 Baseline PUF Performance Evaluation

In our experiment, each chip contains an 8×64 array. The address sets of two adjacent rows are used as challenges, and the 64 pairs of devices in the two rows produce a 64-bit response, by comparing the conductance of the two devices in the corresponding rows (Figure 6.4(b)). Hence, 7 CRPs with 64-bit length can be extracted from one chip. 21 CRPs from 3 chips were used to evaluate the PUF performance. The uniqueness of a PUF represents how distinct of a response comparing to others. It is measured by evaluating the fractional inter Hamming distance (HD)

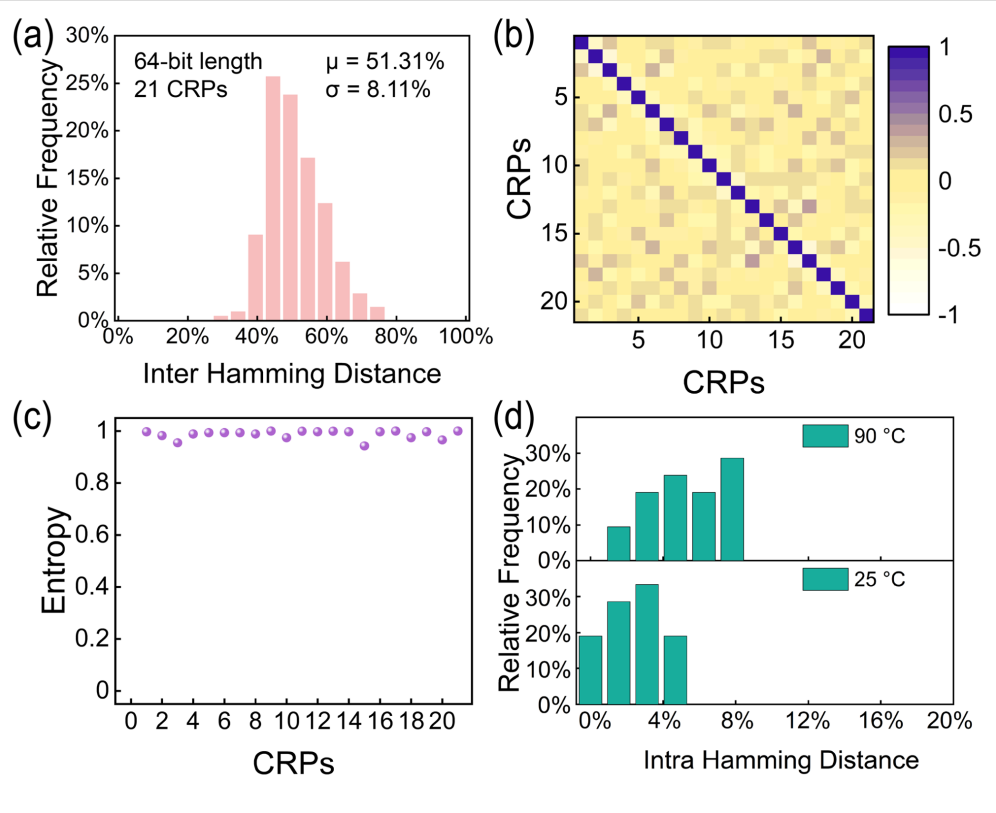


Figure 6.6: (a) Histogram of inter-HD. (b) Correlation matrix of all CRPs. (c) Entropy of 21 CRPs. (d) Intra-HD at 25 °C and 90 °C.

among a group of CRPs. The ideal value of inter-HD is 50% and it is calculated by Equation 6.1, where HD(r_i, r_j) is the HD of i-th and j-th response, and m is the length of response strings:

$$\text{inter-HD} = \frac{2}{n(n+1)} \sum_{i=j+1}^{n} \sum_{j=1}^{n-1} \frac{\text{HD}(r_i, r_j)}{m} \times 100\%. \tag{6.1}$$

Figure 6.6(a) shows the probability density function of inter-HD among 210 (C_2^{21}) pairs of CRPs. The mean value of inter-HD is 51%, corresponding to a nearly ideal uniqueness. Figure 6.6(b) shows the correlation matrix of these CRPs. Only the value on the diagonal entries of the matrix is 1, and the others are all close to 0, suggesting no correlation between different responses.

Entropy is the metric to measure the unpredictability of responses. It follows Equation 6.2, where p represents the probability of finding 1s in the binary bit stream:

$$E = -\left[p \log_2 p + (1-p) \log_2 (1-p)\right]. \tag{6.2}$$

The optimal probability of finding 1s and 0s in a response should be 50%, and the ideal value of entropy is 1. Figure 6.6(c) shows the entropy of 21 CRPs. All of them are close to the ideal value of 1, verifying random bits generated by the fingerprint PUF are unbiased.

We evaluated the reliability of the PUF at both room temperature and elevated temperature. The same challenges were used to obtain responses from the same PUF in different trials. The reliability is evaluated by calculating the intra-HD at different trials. Figure 6.6(d) shows the intra-HD at 25 °C and 90 °C. At 25 °C, the average intra-HD is 2.4%, close to the ideal value 0. High temperature reliability was then measured after the chip was baked at 90 °C for 30 minutes. The average intra-HD at 90 °C degrades to 5.3%, which can be explained by the decrease of the TiO_x film resistance at elevated temperatures, so the measurements are more affected by parasitics including contact resistance and line resistance. We expect the performance will be improved by integrating the PUF with sensing circuitry to minimize the parasitics. The bit fluctuation can be additionally compensated by temporal majority voting, masking vulnerable bits or applying other error correction code [211][217][218]. Better control in the fabrication process that improves the film uniformity will further improve the PUF behavior at both room temperature and elevated temperatures.

Key results of the fingerprint PUF and comparison with prior studies are listed in Table 6.1. Our fingerprint PUF achieves almost desired uniqueness (Inter-HD close to 0.5) and entropy (close to 1). Intra-HD of our PUF is also comparable with other's results at both room and elevated temperature. Compared to other works integrated with advanced technology node, our fingerprint PUF has more simple structure, low-cost fabrication process, and compatible with back-end-of-line process. Besides, our approach does not require any additional stabilization strategies, e.g. error correction code in SRAM PUF, SET/RESET for splitting resistance in RRAM PUF.

Table 6.1: Comparison of different PUF designs.

	Technology	Inter-HD	Intra-HD	Intra-HD (HT)	Entropy	ID Length
SRAM [219]	130 nm	0.6470	0.0304	N/A	N/A	128
RRAM [220]	130 nm	0.4999	0	0 (150 °C)	0.9999	128
RRAM [221]	N/A	0.51	0.0122	0.0593 (90 °C)	N/A	64
Inverter [220]	65 nm	0.4998	0.0300	~0.045 (125 °C)	0.9998	128
MRAM [222]	N/A	0.47	0.0225	N/A	N/A	64
Graphene FET [207]	N/A	0.47	< 0.07	~0.07	N/A	64
This Work [197]	N/A	0.51	0.024	0.053 (90 °C)	0.987	64

6.3.2 Dual Mode Function

By adjusting the via size with respect to the polymer pattern size, a different distribution can be obtained using the same fabrication process. For example, we decreased the via diameter to 1 μm and measured the conductance of 350 devices, as shown in Figure 6.7(a). In this case, almost 50% of the devices are insulating due to the via completely misses the TiO_x region. The conductance of the conductive ones still shows a large variation due to variations of the TiO_x region area within the via. This distribution allows us to use a different approach to build PUF, as shown in Figure 6.7(b). Unlike the differential mode, each device is compared with a pre-determined reference current of 0.9 μS equivalent conductance. The output bit is either 1 or 0, based on whether the read current is higher than the reference current.

Based on this concept, we implemented 20 PUF instantiates with 16-bit length. We evaluated the uniqueness of the PUF by examining the inter-HD and the correlation matrix, which are shown in Figure 6.7(c) and Figure 6.7(d), respectively. The inter-HD achieves a mean value of 51%. In

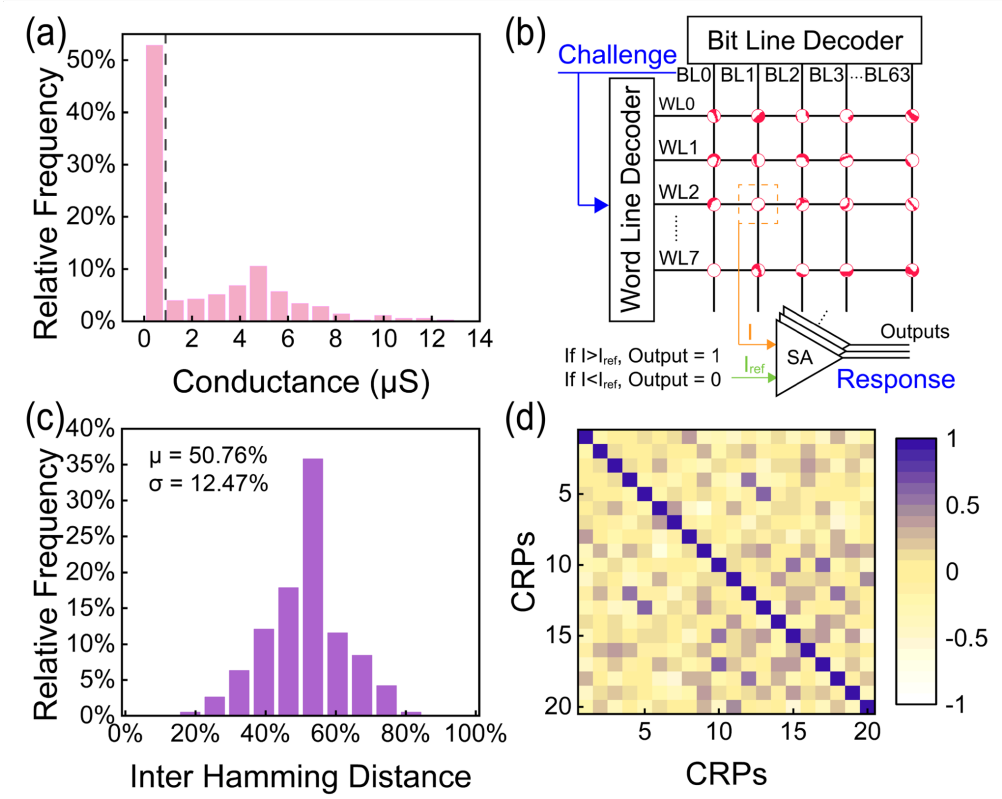


Figure 6.7: (a) Histogram of conductance extracted from 350 devices with 1 μm via. (b) Schematic of PUF system based on the on/off mode. (c) Histogram of inter-HD among CRPs. (d) Correlation matrix of all CRPs.

the correlation matrix, except the entries in diagonal, most other entries are close to 0. Therefore, the on/off mode fingerprint PUF also meets the uniqueness requirement.

6.4 Simulation Analysis

To get more thorough understanding of bit-error rate (BER) and how the reliability can be improved, we studied the BER distribution and how masking vulnerable bits can improve the reliability by simulation. The masked cells are selected from ones with minimum difference in read out current values. The parameters of the simulation are set up by the area distribution of the measured fingerprint patterns. We further assumed different noise levels in the measurement. We simulated 1024 64-bit length responses with 10000 measurements at different noise levels with

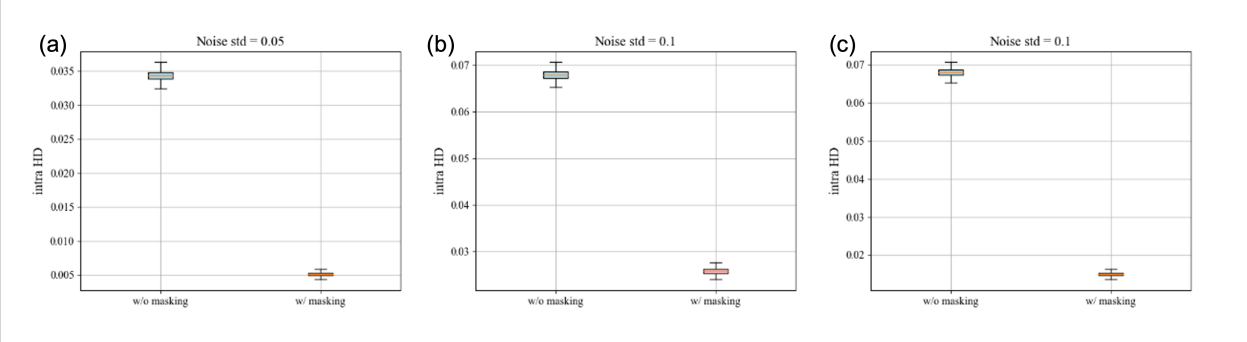


Figure 6.8: Masking syndrome bits to enhance bit-error rate (BER). (a) Noise std = 0.05, masking 8 bits. (b) Noise std = 0.1, masking 8 bits. (c) Noise std = 0.1, masking 12 bits.

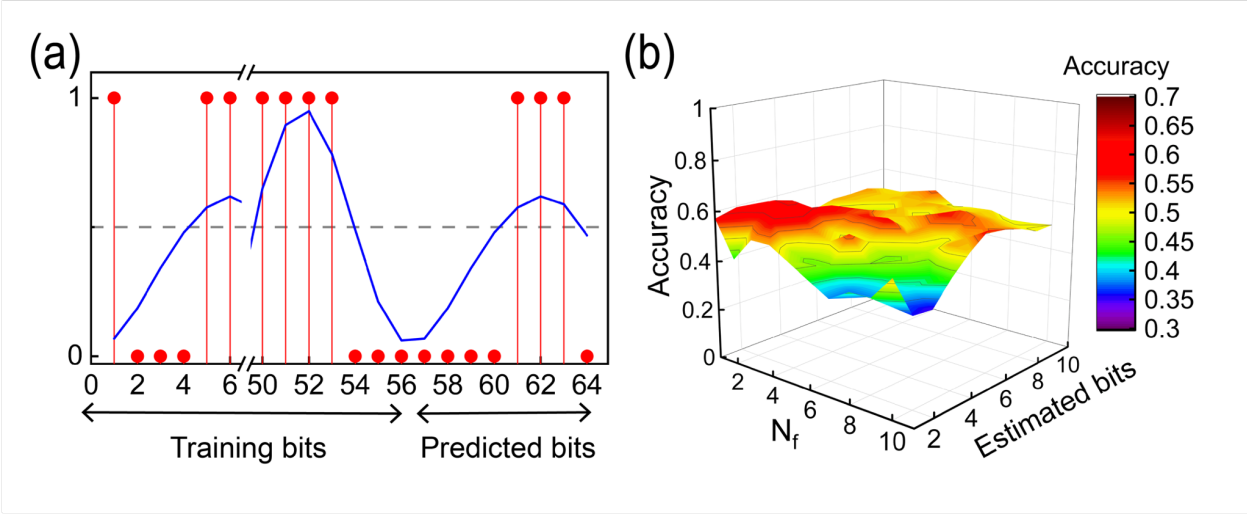


Figure 6.9: (a) Fourier extrapolation prediction on 8 bits. (b) Color map showing prediction accuracy vs. N_f and predicted bits.

standard deviation 0.05 and 0.1. As shown in Figure 6.8(a) when the noise standard deviation is 0.05, the BER of fingerprint PUF maintains a low average value below 0.035. With masking 8 syndrome bits, the BER will be lowered to 0.005. When the noise level increased to 0.1 standard deviation, BER will increased to 0.068 accordingly. As shown in Figure 6.8(b) and (c) by masking out 8 and 12 syndrome bits, BER will decreased to 0.025 and 0.015, respectively. Therefore, by properly choosing the number of masked bits, the BER can be decreased to lower than 0.02 even at large noise levels.

At last, we proved our fingerprint PUF against machine learning attack by implementing a

Fourier extrapolation algorithm with different training bits and Fourier series order N_f. Fourier extrapolation is an effective technique to predict periodic data. If the self-assembled polymer pattern is periodic instead of fully random, then the latter bits can be predicted from the previous ones. The equation used for Fourier extrapolation is shown in Equation 6.3, $f(t)$ is the estimation function that is based on each binary bit of a CRP generated by fingerprint PUF.

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{N_f} (a_n \cos 2\pi kt + b_n \sin 2\pi kt). \tag{6.3}$$

Figure 6.9(a) shows an example of the prediction. The parameters in the estimation function are determined by the training bits. The function will then be extrapolated to predict the predicted bits. We examined the prediction accuracy with Fourier series order from 1 to 10, as well as the number of predicted bits from 1 to 10. Figure 6.9(b) shows the color map of prediction accuracy, which lies between 35–65%. In most scenarios, the prediction accuracy is close to 50%, meaning no better than random guess.

6.5 Conclusion

In this chapter, we show that the intrinsic randomness in fingerprint-like patterns from PS/PMMA polymer phase separation can be used as an entropy source for PUF system design. With properly designed device structure, different conductance distributions can be obtained and used for PUF implementation. The fingerprint PUF achieves excellent uniqueness, entropy, and reliability, and is secure enough to defend machine learning attacks.

CHAPTER 7

Summary and Future Work

7.1 Performance Analysis of DNN Accelerators

In this dissertation, performance modeling and power modeling are done with event-driven, cycle-accurate simulators. These simulators consider the full system of the architecture, and abstract each sub-module as state machine. The simulators are driven by events of each sub-module. All events, such as data forwarding and execution will be recorded with an associated latency. The future plans will further optimize simulators for PIM architectures based on DRAM and RRAM. The optimization will cover defining, architecting, designing, implementing and deploying bit-accurate, cycle-accurate transaction level simulators.

As for DNN accelerators, early-stage simulation provides a systematic methodology of understanding the performance improvement and limitation for DNN accelerators as a function of specific characteristics of the workload and architecture design [223]. Therefore, an architecture simulator should consider the crucial metrics of DNN accelerators. Key metrics of DNN accelerators are listed and analyzed below, which are not constrained to PIM architectures only.

- **Accuracy:** The quality of how well the accelerator performs on certain tasks. Besides tasks under test, a DNN accelerator should be designed with redundancy and reconfigurability to support emerging and more complex models. Accuracy can also represent arithmetic accurate with different data precision, which means it needs to support various data type such as BF16, TF32, FP16, INT8, etc.

- **Latency:** We expect to lower the latency for inference tasks and interactive application. Latency can be optimized by optimizing bandwidth and hiding memory access latency.
- **Throughput:** For high volume data with large batch size, throughput is an important metric at both training and inference. Optimizing peak computation capability or adding more PEs can not improve the throughput solely. We should also maximize the PE utilization rate.
- **Power/Energy:** Power and energy are important for both IoT devices and servers due to the limited battery and cooling capability. Adopting low power design methodology at simulation phase can greatly help analyzing.
- **Flexibility/Scalability:** At the software level, the DNN accelerator should support multiple machine learning frameworks for easy development. At the hardware level, the DNN accelerator should offer scalability at different platforms.
- **Cost:** Besides architecture and chip design cost, cost at system level integration should also be considered. In most cases, this is not required to be covered in the simulator.

The simulators developed in this thesis are flexible for adding new features besides supporting baseline benchmark analysis. Some features that can potentially improve the functionality for future studies are summarized below. 1) Support neural networks with sparse data and model pruning. 2) Allow asynchronous communication among sub-modules and dynamic voltage and frequency scaling. 3) Optimize control flow to reduce instruction overhead. 4) Add more PEs to improve peak computation capability and increase memory bandwidth using more advanced technology node and memory technology. 5) Distribute workload to balance computation and memory, further improve PE utilization rate.

In this dissertation, the skeleton of the PIM simulators for DRAM and RRAM are implemented and improvements will be made the future. The future work will mainly be in three directions: 1) develop simulators to support PIM architectures based on other type of memory, such as 3D NAND flash; 2) implement more functions to the simulator and 3) analyze the event results from the simulation. The detailed future works are summarized as follows.

- **New Memory:** Adopt the feature of event-driven simulators to support PIM architectures with other memory. Besides using the simulation framework for event recording, dataflow, computation logic and weight value mapping should also be tailored based on different memory features.

- **Digital Logic:** Implement simulation functions for digital components for control and data communication. For example, the simulator can have digital logic to support machine learning model advancement such as sparse data processing and model pruning.

- **Model Loading Automation:** Load various pre-trained DNN models directly from PyTorch model files with the specified system configuration. A plan for it is to use PyTorch C++ API to load model weight and model connection diagram. This will be helpful for develop PIM architectures for general machine learn applications.

- **Event Data Analysis:** Dynamic event data can provide many detailed information in hardware architecture design. Automatic analytical methods will make heterogeneous system design more efficient.

7.2 Hardware/Software Co-design of Transformer Accelerator

In Chapter 3, we propose a DRAM-based PIM architecture for GPT acceleration. Besides the autoregressive generation stage, where tokens are generated one by one, a summarizing stage that processes the user's input is also needed. In this step, the model typically processes the input by matrix multiplication in the attention layers. Similarly, encoder-only Transformers such as BERT [97] process the entire sentence at once. PIM-GPT is optimized for decoder-only Transformers. In the future work, we will optimize the system, architecture, dataflow and mapping scheme to support both encoder and decoder architectures. A potential direction is exploiting high storage capability of 3D NAND flash for LLM mapping and tiling. Hence, more diverse models inherited from Transformer are expected to be supported in the future. As for hardware optimization, the dataflow

and mapping scheme of PIM-GPT are inspiring for future architecture development. The ideas of improving computation parallelism and utilizing data locality are transferable to other Transformer accelerators. The future work can take advantage of memory chips with higher storage capability to support larger Transformer models. Moreover, if the memory capacity is sufficient, weight matrices can have multiple copies to support computation parallelism.

Besides hardware optimization, optimizing the Transformer model and algorithm can also boost the performance of accelerators. LinearTransformer [128] developed a linearized attention, which express the self-attention as a linear dot-product of kernel feature maps. By doing so, the memory complexity is reduced from $O(N^2)$ to $O(N)$, where N is the sequence length. Hence, it offers great potential to accelerate very long sequences. Attention Free Transformer [224] does not need to compute and store the expensive attention matrix, instead the result is multiplied in an element-wise fashion. The memory complexity of this operation is linear with respect to both the context size and the dimension of features, rendering it compatible with both large input and model sizes. The removal of FFN layers in the decoder and the utilization of a shared FFN across the encoder in [225] significantly reduces the number of parameters, leading to substantial gains in computational efficiency with only a marginal decrease in accuracy. These approaches can be potentially adopted for Transformer inference acceleration. Moreover, recent studies have shown that the size of Transformer models can be greatly reduced by quantizing the weight to 8-bit [226], 4-bit [227] or even binary [228] without significant degradation in accuracy. The quantization approach can be used to reduce the computation and memory overheads. Redesign floating point arithmetic unit to integer counterpart can also alleviate area and power overheads.

As a summary, the future work on PIM architectures for Transformer beyond PIM-GPT can target on the optimization at hardware level, model level and data level. At hardware level, using memory technology with higher capacity can load and store larger models. At model level, adopting Transformer model optimization can greatly reduce the required computation and memory resources. At data level, training the Transformer model with quantized data can also make the computation more efficient.

7.3 Security of PIM and Neuromorphic Computing System

In Chapter 4 and 5, security vulnerability of 1) model extraction attack and 2) privacy breach are investigated on RRAM-based PIM architecture. In this thesis, we mainly focus on the privacy concerns of PIM architecture for machine learning inference. However, there are also concerns on trustworthiness of the machine learning applications, which can also be used to attack PIM architectures. Attacks targeting trustworthiness aim to influence model behavior during inference or training phases by manipulating or inserting input data, potentially causing malfunctions during inference or the evolving of a compromised model during training. Such attacks are pervasive across various machine learning systems. These types of threats affect the accuracy of models, and can be applied at both inference and training phases. At inference, input data are forwarded to PIM system from sensors or data memory, after necessary preprocessing. Adversaries attempt to deceive the machine learning systems by perturbing the input data or pre-processing pipeline. The goal of adversaries is to make the perturbations imperceptible to system users, but cause the production of incorrect outputs. At training, adversaries have more options to poison the dataset, such as injecting malicious data or manipulating labels. The objective of the adversaries is to generate vulnerable machine learning systems for inference users. Adversarial attacks and poisoning attacks are most representative attacks on machine learning models at inference and training, respectively. The future work on security can analyze the vulnerabilities in the trustworthiness of PIM architectures and study the mitigation strategies to overcome them.

Besides PIM architectures, RRAMs are also used in a wide range of emerging neuromorphic computing schemes, such as spiking neural networks (SNNs) and reservoir computing [229][230][231]. Drawing inspiration from the human brain, recent advancements in neuromorphic computing technologies have emerged as viable alternatives to address the power and latency limitations inherent in traditional digital computing. The implementation of sparse activations effectively minimizes data movement within and beyond the chip, thereby accelerating neuromorphic workloads and resulting in significant gains in both power efficiency and latency when compared to analogous tasks executed on conventional hardware [232]. Given the fundamental difference of

SNN and conventional neural networks, SNNs may have risks in unaware vulnerabilities.

However, due to the involved sparsity feature, low power side-channel leakage and reduced bus access, the security of the neuromorphic computing systems have not been evaluated. Hence, methodically analyzing vulnerabilities of the neuromorphic computing systems from the device, circuit and architecture levels is of high importance and value to the neuromorphic computing community. Potential directions include 1) adversarial attacks on neuromorphic systems, 2) identifying privacy concerns and designing privacy-preserving techniques, 3) investigating vulnerabilities related to the hardware components of neuromorphic chips, 4) modeling side-channel leakage and implementing countermeasures. The research on vulnerability can in turn provide a guideline for designing more reliable neuromorphic computing systems.

BIBLIOGRAPHY

[1] Stephen W Keckler, William J Dally, Brucek Khailany, Michael Garland, and David Glasco. Gpus and the future of parallel computing. *IEEE micro*, 31(5):7–17, 2011.

[2] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, et al. Hardware architecture and software stack for pim based on commercial dram technology: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 43–56. IEEE, 2021.

[3] Onur Mutlu. Memory-centric computing. *arXiv preprint arXiv:2305.20000*, 2023.

[4] Mohammed A Zidan, John Paul Strachan, and Wei D Lu. The future of electronics based on memristive systems. *Nature electronics*, 1(1):22–29, 2018.

[5] Juan Gómez-Luna, Yuxin Guo, Sylvan Brocard, Julien Legriel, Remy Cimadomo, Geraldo F Oliveira, Gagandeep Singh, and Onur Mutlu. Evaluating machine learningworkloads on memory-centric computing systems. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 35–49. IEEE, 2023.

[6] Xinfeng Xie, Zheng Liang, Peng Gu, Abanti Basak, Lei Deng, Ling Liang, Xing Hu, and Yuan Xie. Spacea: Sparse matrix vector multiplication on processing-in-memory accelerator. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 570–583. IEEE, 2021.

[7] Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, et al. Sisa: Set-centric instruction set architecture for graph mining on processing-in-memory systems. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 282–297, 2021.

[8] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 105–117, 2015.

[9] Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, and Onur Mutlu. Genpip: In-memory acceleration of genome analysis via tight integration of basecalling and read mapping. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 710–726. IEEE, 2022.

[10] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. A modern primer on processing in memory. In *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*, pages 171–243. Springer, 2022.

[11] William H Kautz. Cellular logic-in-memory arrays. *IEEE Transactions on Computers*, 100(8):719–727, 1969.

[12] Harold S Stone. A logic-in-memory computer. *IEEE Transactions on Computers*, 100(1):73–78, 1970.

[13] Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, et al. A 1ynm 1.25 v 8gb, 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep-learning applications. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022.

[14] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, Sang-Hyuk Kwon, Je Min Ryu, Jong-Pil Son, O Seongil, Hak-Soo Yu, Haesuk Lee, Soo Young Kim, et al. 25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2 tflops programmable computing unit using bank-level parallelism, for machine learning applications. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 350–352. IEEE, 2021.

[15] Daehan Kwon, Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gi-Moon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, et al. A 1ynm 1.25 v 8gb 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep learning application. *IEEE Journal of Solid-State Circuits*, 58(1):291–302, 2022.

[16] Fabrice Devaux. The true processing in memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–24. IEEE Computer Society, 2019.

[17] Mario Lanza, Abu Sebastian, Wei D Lu, Manuel Le Gallo, Meng-Fan Chang, Deji Akinwande, Francesco M Puglisi, Husam N Alshareef, Ming Liu, and Juan B Roldan. Memristive technologies for data storage, computation, encryption, and radio-frequency communication. *Science*, 376(6597):eabj9979, 2022.

[18] Qiumeng Wei, Bin Gao, Jianshi Tang, He Qian, and Huaqiang Wu. Emerging memory-based chip development for neuromorphic computing: Status, challenges, and perspectives. *IEEE Electron Devices Magazine*, 1(2):33–49, 2023.

[19] H-S Philip Wong and Sayeef Salahuddin. Memory leads the way to better computing. *Nature nanotechnology*, 10(3):191–194, 2015.

[20] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1071–1085. IEEE, 2022.

[21] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, et al. Near-memory processing in action: Accelerating personalized recommendation with axdimm. *IEEE Micro*, 42(1):116–127, 2021.

[22] Seunghwan Cho, Haerang Choi, Eunhyeok Park, Hyunsung Shin, and Sungjoo Yoo. Mcdram v2: In-dynamic random access memory systolic array accelerator to address the large model problem in deep neural networks on the edge. *IEEE Access*, 8:135223–135243, 2020.

[23] Dimin Niu, Shuangchen Li, Yuhao Wang, Wei Han, Zhe Zhang, Yijin Guan, Tianchan Guan, Fei Sun, Fei Xue, Lide Duan, et al. 184qps/w 64mb/mm 2 3d logic-to-dram hybrid bonding with process-near-memory engine for recommendation system. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022.

[24] Alain Denzler, Geraldo F Oliveira, Nastaran Hajinazar, Rahul Bera, Gagandeep Singh, Juan Gómez-Luna, and Onur Mutlu. Casper: Accelerating stencil computations using near-cache processing. *IEEE Access*, 11:22136–22154, 2023.

[25] Gunjae Koo, Kiran Kumar Matam, Te I, HV Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. Summarizer: trading communication with computing near storage. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 219–231, 2017.

[26] Y Wu, F Cai, L Thomas, T Liu, A Nourbakhsh, J Hebding, E Smith, R Quon, R Smith, A Kumar, et al. Demonstration of a multi-level μa-range bulk switching reram and its application for keyword spotting. In *2022 International Electron Devices Meeting (IEDM)*, pages 18–4. IEEE, 2022.

[27] Yuting Wu, Qiwen Wang, Ziyu Wang, Xinxin Wang, Buvna Ayyagari, Siddarth Krishnan, Michael Chudzik, and Wei D Lu. Bulk-switching memristor-based compute-in-memory module for deep neural network training. *arXiv preprint arXiv:2305.14547*, 2023.

[28] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Compute caches. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–492. IEEE, 2017.

[29] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *2018 ACM/IEEE 45Th annual international symposium on computer architecture (ISCA)*, pages 383–396. IEEE, 2018.

[30] Minki Jeong and Wanyeong Jung. Mac-do: Charge based multi-bit analog in-memory accelerator compatible with dram using output stationary mapping. *arXiv preprint arXiv:2207.07862*, 2022.

[31] Hunjun Lee, Minseop Kim, Dongmoon Min, Joonsung Kim, Jongwon Back, Honam Yoo, Jong-Ho Lee, and Jangwoo Kim. 3d-fpim: An extreme energy-efficient dnn acceleration system using 3d nand flash-based in-situ pim unit. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1359–1376. IEEE, 2022.

[32] Yuting Wu, Ziyu Wang, and Wei D Lu. Pim-gpt: A hybrid process-in-memory accelerator for autoregressive transformers. *arXiv preprint arXiv:2310.09385*, 2023.

[33] Ziyu Wang, Fan-hsuan Meng, Yongmo Park, Jason K Eshraghian, and Wei D Lu. Side-channel attack analysis on in-memory computing architectures. *IEEE Transactions on Emerging Topics in Computing*, 2023.

[34] Ziyu Wang, Yuting Wu, Yongmo Park, Sangmin Yoo, Xinxin Wang, Jason K Eshraghian, and Wei D Lu. Powergan: A machine learning approach for power side-channel attack on compute-in-memory accelerators. *arXiv preprint arXiv:2304.11056*, 2023.

[35] Yongmo Park, Ziyu Wang, Sangmin Yoo, and Wei D Lu. Rm-ntt: An rram-based compute-in-memory number theoretic transform accelerator. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 8(2):93–101, 2022.

[36] Yuting Wu, Xinxin Wang, and Wei D Lu. Dynamic resistive switching devices for neuromorphic computing. *Semiconductor Science and Technology*, 37(2):024003, 2021.

[37] Suhas Kumar, Xinxin Wang, John Paul Strachan, Yuchao Yang, and Wei D Lu. Dynamical memristors for higher-complexity neuromorphic computing. *Nature Reviews Materials*, 7(7):575–591, 2022.

[38] John Moon, Yuting Wu, and Wei D Lu. Hierarchical architectures in reservoir computing systems. *Neuromorphic Computing and Engineering*, 1(1):014006, 2021.

[39] Chao Du, Fuxi Cai, Mohammed A Zidan, Wen Ma, Seung Hwan Lee, and Wei D Lu. Reservoir computing using dynamic memristors for temporal information processing. *Nature communications*, 8(1):2204, 2017.

[40] Qing Dong, Mahmut E Sinangil, Burak Erbagci, Dar Sun, Win-San Khwa, Hung-Jen Liao, Yih Wang, and Jonathan Chang. 15.3 a 351tops/w and 372.4 gops compute-in-memory sram macro in 7nm finfet cmos for machine-learning applications. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 242–244. IEEE, 2020.

[41] Mahmut E Sinangil, Burak Erbagci, Rawan Naous, Kerem Akarvardar, Dar Sun, Win-San Khwa, Hung-Jen Liao, Yih Wang, and Jonathan Chang. 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*, 56(1):188–198, 2020.

[42] Jack Choquette and Wish Gandhi. Nvidia a100 gpu: Performance & innovation for gpu computing. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–43. IEEE Computer Society, 2020.

[43] 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. IEEE, 2021.

[44] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.

[45] Zhe Zhou, Cong Li, Fan Yang, and Guangyu Suny. Dimm-link: Enabling efficient inter-dimm communication for near-memory processing. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 302–316. IEEE, 2023.

[46] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A case for exploiting subarray-level parallelism (salp) in dram. *ACM SIGARCH Computer Architecture News*, 40(3):368–379, 2012.

[47] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. Drisa: A dram-based reconfigurable in-situ accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 288–301, 2017.

[48] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. Computedram: In-memory compute using off-the-shelf drams. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, pages 100–113, 2019.

[49] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F Oliveira, and Onur Mutlu. Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system. *IEEE Access*, 10:52565–52608, 2022.

[50] Yuting Wu, John Moon, Xiaojian Zhu, and Wei D Lu. Neural functional connectivity reconstruction with second-order memristor network. *Advanced Intelligent Systems*, 3(8):2000276, 2021.

[51] John Moon, Yuting Wu, Xiaojian Zhu, and Wei D Lu. Neural connectivity inference with spike-timing dependent plasticity network. *Science China Information Sciences*, 64(6):160405, 2021.

[52] Yuting Wu, Sangmin Yoo, Fan-Hsuan Meng, and Wei D Lu. Spatiotemporal spike pattern detection with second-order memristive synapses. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 625–628. IEEE, 2022.

[53] J Joshua Yang, Matthew D Pickett, Xuema Li, Douglas AA Ohlberg, Duncan R Stewart, and R Stanley Williams. Memristive switching mechanism for metal/oxide/metal nanodevices. *Nature nanotechnology*, 3(7):429–433, 2008.

[54] Yuchao Yang, ShinHyun Choi, and Wei Lu. Oxide heterostructure resistive memory. *Nano letters*, 13(6):2908–2915, 2013.

[55] Yuchao Yang, Peng Gao, Siddharth Gaba, Ting Chang, Xiaoqing Pan, and Wei Lu. Observation of conducting filament growth in nanoscale resistive memories. *Nature communications*, 3(1):732, 2012.

[56] YM Sun, C Song, J Yin, LL Qiao, R Wang, ZY Wang, XZ Chen, SQ Yin, MS Saleem, HQ Wu, et al. Modulating metallic conductive filaments via bilayer oxides in resistive switching memory. *Applied Physics Letters*, 114(19), 2019.

[57] Ting Chang, Sung-Hyun Jo, and Wei Lu. Short-term memory to long-term memory transition in a nanoscale memristor. *ACS nano*, 5(9):7669–7676, 2011.

[58] Xiaojian Zhu, Jihang Lee, and Wei D Lu. Iodine vacancy redistribution in organic–inorganic halide perovskite films and resistive switching effects. *Advanced Materials*, 29(29):1700527, 2017.

[59] Yiming Sun, Meiqian Tai, Cheng Song, Ziyu Wang, Jun Yin, Fan Li, Huaqiang Wu, Fei Zeng, Hong Lin, and Feng Pan. Competition between metallic and vacancy defect conductive filaments in a ch3nh3pbi3-based memory device. *The Journal of Physical Chemistry C*, 122(11):6431–6436, 2018.

[60] Ziyu Wang, Yiming Sun, Xiaofeng Zhou, Feng Pan, and Cheng Song. Local control of exchange bias by resistive switching. *physica status solidi (RRL)–Rapid Research Letters*, 12(12):1800446, 2018.

[61] Chih-Yang Lin, Jia Chen, Po-Hsun Chen, Ting-Chang Chang, Yuting Wu, Jason K Eshraghian, John Moon, Sangmin Yoo, Yu-Hsun Wang, Wen-Chung Chen, et al. Adaptive synaptic memory via lithium ion modulation in rram devices. *Small*, 16(42):2003964, 2020.

[62] Andrew D Kent and Daniel C Worledge. A new spin on magnetic memories. *Nature nanotechnology*, 10(3):187–191, 2015.

[63] Geoffrey W Burr, Matthew J Breitwisch, Michele Franceschini, Davide Garetto, Kailash Gopalakrishnan, Bryan Jackson, Bülent Kurdi, Chung Lam, Luis A Lastras, Alvaro Padilla, et al. Phase change memory technology. *Journal of Vacuum Science & Technology B*, 28(2):223–262, 2010.

[64] Giorgio Servalli. A 45nm generation phase change memory technology. In *2009 IEEE international electron devices meeting (IEDM)*, pages 1–4. IEEE, 2009.

[65] H Mulaosmanovic, J Ocker, S Müller, M Noack, J Müller, P Polakowski, T Mikolajick, and S Slesazeck. Novel ferroelectric fet based synapse for neuromorphic systems. In *2017 Symposium on VLSI Technology*, pages T176–T177. IEEE, 2017.

[66] Stefan Dünkel, Martin Trentzsch, Regina Richter, Patrick Moll, Christine Fuchs, Oliver Gehring, Mateusz Majer, Stefan Wittek, B Müller, Thomas Melde, et al. A fefet based super-low-power ultra-fast embedded nvm technology for 22nm fdsoi and beyond. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 19–7. IEEE, 2017.

[67] Jianshi Tang, Douglas Bishop, Seyoung Kim, Matt Copel, Tayfun Gokmen, Teodor Todorov, SangHoon Shin, Ko-Tao Lee, Paul Solomon, Kevin Chan, et al. Ecram as scalable synaptic cell for high-speed, low-power neuromorphic computing. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 13–1. IEEE, 2018.

[68] Jinsong Cui, Fufei An, Jiangchao Qian, Yuxuan Wu, Luke L Sloan, Saran Pidaparthy, Jian-Min Zuo, and Qing Cao. Cmos-compatible electrochemical synaptic transistor arrays for deep learning accelerators. *Nature Electronics*, 6(4):292–300, 2023.

[69] Tianyu Wang, Jialin Meng, Xufeng Zhou, Yue Liu, Zhenyu He, Qi Han, Qingxuan Li, Jiajie Yu, Zhenhai Li, Yongkai Liu, et al. Reconfigurable neuromorphic memristor network for ultralow-power smart textile electronics. *Nature Communications*, 13(1):7432, 2022.

[70] Yiming Sun, Xiaolong Zhao, Cheng Song, Kun Xu, Yue Xi, Jun Yin, Ziyu Wang, Xiaofeng Zhou, Xianzhe Chen, Guoyi Shi, et al. Performance-enhancing selector via symmetrical multilayer design. *Advanced Functional Materials*, 29(13):1808376, 2019.

[71] Xiaoxin Xu, Qing Luo, Tiancheng Gong, Hangbing Lv, Shibing Long, Qi Liu, Steve S Chung, Jing Li, and Ming Liu. Fully cmos compatible 3d vertical rram with self-aligned self-selective cell enabling sub-5nm scaling. In *2016 IEEE Symposium on VLSI Technology*, pages 1–2. IEEE, 2016.

[72] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[73] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301, 2010.

[74] Mirko Prezioso, Farnood Merrikh-Bayat, Brian D Hoskins, Gina C Adam, Konstantin K Likharev, and Dmitri B Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61–64, 2015.

[75] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News*, 44(3):27–39, 2016.

[76] Xinxin Wang, Yuting Wu, and Wei D Lu. Rram-enabled ai accelerator architecture. In *2021 IEEE International Electron Devices Meeting (IEDM)*, pages 12–2. IEEE, 2021.

[77] Je-Min Hung, Chuan-Jia Jhang, Ping-Chun Wu, Yen-Cheng Chiu, and Meng-Fan Chang. Challenges and trends of nonvolatile in-memory-computation circuits for ai edge devices. *IEEE Open Journal of the Solid-State Circuits Society*, 1:171–183, 2021.

[78] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[79] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 385–399, 2020.

[80] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

[81] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang 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*, pages 393–406, 2018.

[82] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[83] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. Open dnn box by power side-channel attack. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(11):2717–2721, 2020.

[84] Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[85] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. Csi nn: Reverse engineering of neural network architectures through electromagnetic side channel. 2019.

[86] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. Stealing neural network structure through remote fpga side-channel analysis. *IEEE Transactions on Information Forensics and Security*, 16:4377–4388, 2021.

[87] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.

[88] Gagandeep Singh, Juan Gómez-Luna, Giovanni Mariani, Geraldo F Oliveira, Stefano Corda, Sander Stuijk, Onur Mutlu, and Henk Corporaal. Napel: Near-memory computing application performance prediction via ensemble learning. In *Proceedings of the 56th annual design automation conference 2019*, pages 1–6, 2019.

[89] Sheng Xu, Xiaoming Chen, Ying Wang, Yinhe Han, Xuehai Qian, and Xiaowei Li. Pimsim: A flexible and detailed processing-in-memory simulator. *IEEE Computer Architecture Letters*, 18(1):6–9, 2018.

[90] Chao Yu, Sihang Liu, and Samira Khan. Multipim: A detailed and configurable multi-stack processing-in-memory simulator. *IEEE Computer Architecture Letters*, 20(1):54–57, 2021.

[91] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(12):3067–3080, 2018.

[92] Sourjya Roy, Shrihari Sridharan, Shubham Jain, and Anand Raghunathan. TxSim: Modeling training of deep neural networks on resistive crossbar systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(4):730–738, 2021.

[93] Lixue Xia, Boxun Li, Tianqi Tang, Peng Gu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, Yuan Xie, and Huazhong Yang. MNSIM: Simulation platform for memristor-based neuromorphic computing system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(5):1009–1022, 2017.

[94] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[95] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer architecture letters*, 15(1):45–49, 2015.

[96] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[97] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[98] R OpenAI. Gpt-4 technical report. *arXiv*, pages 2303–08774, 2023.

[99] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18*, pages 194–206. Springer, 2019.

[100] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit S Dhillon. Taming pretrained transformers for extreme multi-label text classification. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3163–3171, 2020.

[101] Siddhant Garg and Goutham Ramakrishnan. Bae: Bert-based adversarial examples for text classification. *arXiv preprint arXiv:2004.01970*, 2020.

[102] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*, 2019.

[103] Shaowei Yao and Xiaojun Wan. Multimodal transformer for multimodal machine translation. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4346–4350, 2020.

[104] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[105] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[106] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[107] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE, 2021.

[108] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 616–630. IEEE, 2022.

[109] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W Lee, et al. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 328–341. IEEE, 2020.

[110] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W Lee. Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 692–705. IEEE, 2021.

[111] Hanhwi Jang, Joonsung Kim, Jae-Eon Jo, Jaewon Lee, and Jangwoo Kim. Mnnfast: A fast and scalable system architecture for memory-augmented neural networks. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 250–263, 2019.

[112] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas 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)*, pages 811–824. IEEE, 2020.

[113] Haoran Wang, Haobo Xu, Ying Wang, and Yinhe Han. Cta: Hardware-software co-design for compressed token attention mechanism. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 429–441. IEEE, 2023.

[114] Jyotikrishna Dass, Shang Wu, Huihong Shi, Chaojian Li, Zhifan Ye, Zhongfeng Wang, and Yingyan Lin. Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 415–428. IEEE, 2023.

[115] Haoran You, Zhanyi Sun, Huihong Shi, Zhongzhi Yu, Yang Zhao, Yongan Zhang, Chaojian Li, Baopu Li, and Yingyan Lin. Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 273–286. IEEE, 2023.

[116] Saugata Ghose, Amirali Boroumand, Jeremie S Kim, Juan Gómez-Luna, and Onur Mutlu. Processing-in-memory: A workload-driven perspective. *IBM Journal of Research and Development*, 63(6):3–1, 2019.

[117] Hyunsung Shin, Dongyoung Kim, Eunhyeok Park, Sungho Park, Yongsik Park, and Sungjoo Yoo. Mcdram: Low latency and energy-efficient matrix computations in dram. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2613–2622, 2018.

[118] Amir Yazdanbakhsh, Choungki Song, Jacob Sacks, Pejman Lotfi-Kamran, Hadi Esmaeilzadeh, and Nam Sung Kim. In-dram near-data approximate acceleration for gpus. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, pages 1–14, 2018.

[119] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[120] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. Fine-grained dram: Energy-efficient dram for extreme bandwidth systems. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 41–54, 2017.

[121] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and TN Vijaykumar. Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 372–385. IEEE, 2020.

[122] Zichen Fan, Qirui Zhang, Pierre Abillama, Sara Shoouri, Changwoo Lee, David Blaauw, Hun-Seok Kim, and Dennis Sylvester. Taskfusion: An efficient transfer learning architecture with dual delta sparsity for multi-task natural language processing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[123] Fast inverse square root. https://en.wikipedia.org/wiki/fast_inverse_square_root, accessed: July 2023.

[124] Micron. 16gb ddr5 sdram addendum, https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr5/16gb_ddr5_sdram_diereva.pdf, accessed: May 2023.

[125] Micron. Tn-ed-03: Gddr6: The next-generation graphics dram.

[126] Saugata Ghose, Abdullah Giray Yaglikçi, Raghav Gupta, Donghyuk Lee, Kais Kudrolli, William X Liu, Hasan Hassan, Kevin K Chang, Niladrish Chatterjee, Aditya Agrawal, et al. What your dram power models are not telling you: Lessons from a detailed experimental study. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(3):1–41, 2018.

[127] Yongkee Kwon, Kornijcuk Vladimir, Nahsung Kim, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, et al. System architecture and software stack for gddr6-aim. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–25. IEEE, 2022.

[128] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[129] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[130] Y Tay, M Dehghani, D Bahri, and D Metzler. Efficient transformers: A survey. arxiv 2020. *arXiv preprint arXiv:2009.06732*.

[131] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[132] Rui-Jie Zhu, Qihang Zhao, and Jason K Eshraghian. Spikegpt: Generative pre-trained language model with spiking neural networks. *arXiv preprint arXiv:2302.13939*, 2023.

[133] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3):264–274, 2020.

[134] Md Tanvir Arafin and Zhaojun Lu. Security challenges of processing-in-memory systems. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pages 229–234, 2020.

[135] Mohammad Nasim Imtiaz Khan, Shivam Bhasin, Bo Liu, Alex Yuan, Anupam Chattopadhyay, and Swaroop Ghosh. Comprehensive study of side-channel attack on emerging non-volatile memories. *Journal of Low Power Electronics and Applications*, 11(4):38, 2021.

[136] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[137] Fan Zhang and Miao Hu. CCCS: customized spice-level crossbar-array circuit simulator for in-memory computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–8, 2020.

[138] Corey Lammie, Wei Xiang, Bernabé Linares-Barranco, and Mostafa Rahimi Azghadi. MemTorch: An open-source simulation framework for memristive deep learning systems. *Neurocomputing*, 485:124–133, 2022.

[139] Malte J Rasch, Diego Moreda, Tayfun Gokmen, Manuel Le Gallo, Fabio Carta, Cindy Goldberg, Kaoutar El Maghraoui, Abu Sebastian, and Vijay Narayanan. A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays. In *2021 IEEE 3rd international conference on artificial intelligence circuits and systems (AICAS)*, pages 1–4. IEEE, 2021.

[140] Xiaochen Peng, Shanshi Huang, Yandong Luo, Xiaoyu Sun, and Shimeng Yu. DNN+ neuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. In *2019 IEEE international electron devices meeting (IEDM)*, pages 32–5. IEEE, 2019.

[141] Wei-Ting Lin, Hsiang-Yun Cheng, Chia-Lin Yang, Meng-Yao Lin, Kai Lien, Han-Wen Hu, Hung-Sheng Chang, Hsiang-Pang Li, Meng-Fan Chang, Yen-Ting Tsou, et al. DL-RSIM: A reliability and deployment strategy simulation framework for ReRAM-based CNN accelerators. *ACM Transactions on Embedded Computing Systems (TECS)*, 2022.

[142] Hyein Shin, Myeonggu Kang, and Lee-Sup Kim. A thermal-aware optimization framework for ReRAM-based deep neural network acceleration. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[143] Majed Valad Beigi and Gokhan Memik. Thermal-aware optimizations of ReRAM-based neuromorphic computing systems. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[144] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[145] Qiwen Wang, Xinxin Wang, Seung Hwan Lee, Fan-Hsuan Meng, and Wei D Lu. A deep neural network accelerator based on tiled RRAM architecture. In *2019 IEEE international electron devices meeting (IEDM)*, pages 14–4. IEEE, 2019.

[146] Daniel Lorenz, Philipp A Hartmann, Kim Grüttner, and Wolfgang Nebel. Non-invasive power simulation at system-level with SystemC. In *International Workshop on Power and Timing Modeling, Optimization and Simulation*, pages 21–31. Springer, 2012.

[147] Brian P Ginsburg and Anantha P Chandrakasan. An energy-efficient charge recycling approach for a SAR converter with capacitive DAC. In *2005 IEEE international symposium on circuits and systems*, pages 184–187. IEEE, 2005.

[148] V Hariprasath, Jon Guerber, S-H Lee, and U-K Moon. Merged capacitor switching based SAR ADC with highest switching energy-efficiency. *Electronics letters*, 46(9):620–621, 2010.

[149] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry 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*, pages 2704–2713, 2018.

[150] Jason K Eshraghian, Corey Lammie, Mostafa Rahimi Azghadi, and Wei D Lu. Navigating local minima in quantized spiking neural networks. *arXiv preprint arXiv:2202.07221*, 2022.

[151] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE international symposium on high performance computer architecture (HPCA)*, pages 541–552. IEEE, 2017.

[152] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of secure ecc implementations: a survey on known side-channel

attacks and countermeasures. In *2010 IEEE international symposium on hardware-oriented security and trust (HOST)*, pages 76–87. IEEE, 2010.

[153] Martin Hutle and Markus Kammerstetter. Resilience against physical attacks. In *Smart Grid Security*, pages 79–112. Elsevier, 2015.

[154] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[155] Weier Wan, Rajkumar Kubendran, Clemens Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, et al. A compute-in-memory chip based on resistive random-access memory. *Nature*, 608(7923):504–512, 2022.

[156] Wantong Li, James Read, Hongwu Jiang, and Shimeng Yu. A 40nm rram compute-in-memory macro with parallelism-preserving ecc for iso-accuracy voltage scaling. In *ESSCIRC 2022-IEEE 48th European Solid State Circuits Conference (ESSCIRC)*, pages 101–104. IEEE, 2022.

[157] Justin M Correll, Lu Jie, Seungheun Song, Seungjong Lee, Junkang Zhu, Wei Tang, Luke Wormald, Jack Erhardt, Nicolas Breil, Roger Quon, et al. An 8-bit 20.7 tops/w multi-level cell reram-based compute engine. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 264–265. IEEE, 2022.

[158] Su-in Yi, Jack D Kendall, R Stanley Williams, and Suhas Kumar. Activity-difference training of deep neural networks using memristor crossbars. *Nature Electronics*, pages 1–7, 2022.

[159] Hiroki Funato and Takashi Suga. Magnetic near-field probe for ghz band and spatial resolution improvement technique. In *2006 17th International Zurich Symposium on Electromagnetic Compatibility*, pages 284–287. IEEE, 2006.

[160] Yien-Tien Chou and Hsin-Chia Lu. Space difference magnetic near-field probe with spatial resolution improvement. *IEEE transactions on microwave theory and techniques*, 61(12):4233–4244, 2013.

[161] ZhiHe Peng, XiaoChun Li, and JunFa Mao. A pair of parallel differential magnetic-field probes with high spatial resolution and wide frequency bandwidth. In *2019 IEEE MTT-S International Wireless Symposium (IWS)*, pages 1–3. IEEE, 2019.

[162] Mohammad Nasim Imtiaz Khan, Shivam Bhasin, Alex Yuan, Anupam Chattopadhyay, and Swaroop Ghosh. Side-channel attack on sttram based cache for cryptographic application. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 33–40. IEEE, 2017.

[163] Keysight. Infiniium uxr-series oscilloscopes data sheet, https://www.keysight.com/zz/en/assets/7018-06242/data-sheets/5992-3132.pdf, accessed: 2010-09-30.

[164] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[165] Chenguang Wang, Ming Yan, Yici Cai, Qiang Zhou, and Jianlei Yang. Power profile equalizer: A lightweight countermeasure against side-channel attack. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 305–312. IEEE, 2017.

[166] Shih-Hao Cheng, Meng-Hsueh Lee, Bing-Chen Wu, and Tsung-Te Liu. A lightweight power side-channel attack protection technique with minimized overheads using on-demand current equalizer. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(10):4008–4012, 2022.

[167] Carlos Tokunaga and David Blaauw. Securing encryption systems with a switched capacitor current equalizer. *IEEE Journal of Solid-State Circuits*, 45(1):23–31, 2009.

[168] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020, 2020.

[169] Ehsan Variani, Xin Lei, Erik McDermott, Ignacio Lopez Moreno, and Javier Gonzalez-Dominguez. Deep neural networks for small footprint text-dependent speaker verification. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4052–4056. IEEE, 2014.

[170] Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. Deep learning for financial applications: A survey. *Applied Soft Computing*, 93:106384, 2020.

[171] Haiyu Mao, Mingcong Song, Tao Li, Yuting Dai, and Jiwu Shu. Lergan: A zero-free, low data movement and pim-based gan architecture. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 669–681. IEEE, 2018.

[172] Fan Chen, Linghao Song, and Yiran Chen. Regan: A pipelined reram-based accelerator for generative adversarial networks. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 178–183. IEEE, 2018.

[173] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577(7792):641–646, 2020.

[174] Zhongrui Wang, Can Li, Peng Lin, Mingyi Rao, Yongyang Nie, Wenhao Song, Qinru Qiu, Yunning Li, Peng Yan, John Paul Strachan, et al. In situ training of feed-forward and recurrent convolutional memristor networks. *Nature Machine Intelligence*, 1(9):434–442, 2019.

[175] Fan-Hsuan Meng, Xinxin Wang, Ziyu Wang, Eric Yeu-Jer Lee, and Wei D Lu. Exploring compute-in-memory architecture granularity for structured pruning of neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 12(4):858–866, 2022.

[176] Ziyun Li, Zhehong Wang, Li Xu, Qing Dong, Bowen Liu, Chin-I Su, Wen-Ting Chu, George Tsou, Yu-Der Chih, Tsung-Yung Jonathan Chang, et al. Rram-dnn: An rram and model-compression empowered all-weights-on-chip dnn accelerator. *IEEE Journal of Solid-State Circuits*, 56(4):1105–1115, 2020.

[177] Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221–248, 2017.

[178] Han Zhao, Zhengwu Liu, Jianshi Tang, Bin Gao, Qi Qin, Jiaming Li, Ying Zhou, Peng Yao, Yue Xi, Yudeng Lin, et al. Energy-efficient high-fidelity image reconstruction with memristor arrays for medical diagnosis. *Nature Communications*, 14(1):2276, 2023.

[179] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[180] Mateusz Buda, Ashirbani Saha, and Maciej A Mazurowski. Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm. *Computers in biology and medicine*, 109:218–225, 2019.

[181] M. Buda. Brain mri segmentation dataset, https://www.kaggle.com/datasets/mateuszbuda/lgg-mri-segmentation.

[182] Ronald Rohrer, Laurence Nagel, Robert Meyer, and Lynn Weber. Computationally efficient electronic-circuit noise calculations. *IEEE Journal of Solid-State Circuits*, 6(4):204–213, 1971.

[183] Richard Schreier, José Silva, Jesper Steensgaard, and Gabor C Temes. Design-oriented estimation of thermal noise in switched-capacitor circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(11):2358–2368, 2005.

[184] Ali Sheikholeslami. Power supply noise [circuit intuitions]. *IEEE Solid-State Circuits Magazine*, 12(3):15–17, 2020.

[185] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys & Tutorials*, 20(1):465–488, 2017.

[186] Debayan Das, Shovan Maity, Saad Bin Nasir, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. High efficiency power side-channel attack immunity using noise injection in attenuated signature domain. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 62–67. IEEE, 2017.

[187] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Computing Surveys*, 55(11):1–35, 2023.

[188] Takaya Kubota, Kota Yoshida, Mitsuru Shiozaki, and Takeshi Fujino. Deep learning side-channel attack against hardware implementations of aes. *Microprocessors and Microsystems*, 87:103383, 2021.

[189] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: a survey. *Journal of Cryptographic Engineering*, 10:135–162, 2020.

[190] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[191] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[192] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *International conference on machine learning*, pages 1060–1069. PMLR, 2016.

[193] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[194] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[195] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[196] Yansong Gao, Said F Al-Sarawi, and Derek Abbott. Physical unclonable functions. *Nature Electronics*, 3(2):81–91, 2020.

[197] Ziyu Wang, Xiaojian Zhu, Supreet Jeloka, Brian Cline, and Wei D Lu. Physical unclonable function systems based on pattern transfer of fingerprint-like patterns. *IEEE Electron Device Letters*, 43(4):655–658, 2022.

[198] Sandip Ray, Eric Peeters, Mark M Tehranipoor, and Swarup Bhunia. System-on-chip platform security assurance: Architecture and validation. *Proceedings of the IEEE*, 106(1):21–37, 2017.

[199] Charles Herder, Meng-Day Yu, Farinaz Koushanfar, and Srinivas Devadas. Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102(8):1126–1141, 2014.

[200] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual design automation conference*, pages 9–14, 2007.

[201] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

[202] Pim Tuyls, Geert-Jan Schrijen, Boris Škorić, Jan Van Geloven, Nynke Verhaegh, and Rob Wolters. Read-proof hardware from protective coatings. In *Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8*, pages 369–383. Springer, 2006.

[203] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 148–160, 2002.

[204] Jorge Guajardo, Sandeep S Kumar, Geert-Jan Schrijen, and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. In *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*, pages 63–80. Springer, 2007.

[205] Mafalda Cortez, Apurva Dargar, Said Hamdioui, and Geert-Jan Schrijen. Modeling sram start-up behavior for physical unclonable functions. In *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2012.

[206] Bohan Lin, Yachuan Pang, Bin Gao, Jianshi Tang, Dong Wu, Ting-Wei Chang, Wei-En Lin, Xiaoyu Sun, Shimeng Yu, Meng-Fan Chang, et al. A highly reliable rram physically unclonable function utilizing post-process randomness source. *IEEE Journal of Solid-State Circuits*, 56(5):1641–1650, 2021.

[207] Akhil Dodda, Shiva Subbulakshmi Radhakrishnan, Thomas F Schranghamer, Drew Buzzell, Parijat Sengupta, and Saptarshi Das. Graphene-based physically unclonable functions that are reconfigurable and resilient to machine learning attacks. *Nature Electronics*, 4(5):364–374, 2021.

[208] An Chen. Comprehensive assessment of rram-based puf for hardware security applications. In *2015 IEEE international electron devices meeting (IEDM)*, pages 10–7. IEEE, 2015.

[209] Le Zhang, Zhi Hui Kong, and Chip-Hong Chang. Pckgen: A phase change memory based cryptographic key generator. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1444–1447. IEEE, 2013.

[210] Jayita Das, Kevin Scott, Srinath Rajaram, Drew Burgett, and Sanjukta Bhanja. Mram puf: A novel geometry based magnetic puf with integrated cmos. *IEEE Transactions on Nanotechnology*, 14(3):436–443, 2015.

[211] Meng-Day Yu and Srinivas Devadas. Secure and robust error correction for physical unclonable functions. *IEEE Design & Test of Computers*, 27(1):48–65, 2010.

[212] Riccardo Bernardini and Roberto Rinaldo. Helper-less physically unclonable functions and chip authentication. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8193–8197. IEEE, 2014.

[213] Jin Miao, Meng Li, Subhendu Roy, and Bei Yu. Lrr-dpuf: Learning resilient and reliable digital physical unclonable function. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2016.

[214] Ryan Helinski, Dhruva Acharyya, and Jim Plusquellic. A physical unclonable function defined using power distribution system equivalent resistance variations. In *Proceedings of the 46th Annual Design Automation Conference*, pages 676–681, 2009.

[215] Stefan Walheim, Martin Böltau, Jürgen Mlynek, Georg Krausch, and Ullrich Steiner. Structure formation via polymer demixing in spin-cast films. *Macromolecules*, 30(17):4995–5003, 1997.

[216] Stefan Walheim, Erik Schaffer, Jurgen Mlynek, and Ullrich Steiner. Nanophase-separated polymer films as high-performance antireflection coatings. *Science*, 283(5401):520–522, 1999.

[217] Jeroen Delvaux, Dawu Gu, Dries Schellekens, and Ingrid Verbauwhede. Helper data algorithms for puf-based key generation: Overview and analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(6):889–902, 2014.

[218] Matthias Hiller, Dominik Merli, Frederic Stumpf, and Georg Sigl. Complementary ibs: Application specific error correction for pufs. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 1–6. IEEE, 2012.

[219] Ying Su, Jeremy Holleman, and Brian Otis. A 1.6 pj/bit 96% stable chip-id generating circuit using process variations. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 406–611. IEEE, 2007.

[220] Yachuan Pang, Bin Gao, Dong Wu, Shengyu Yi, Qi Liu, Wei-Hao Chen, Ting-Wei Chang, Wei-En Lin, Xiaoyu Sun, Shimeng Yu, et al. 25.2 a reconfigurable rram physically unclonable function utilizing post-process randomness source with¡ 6× 10- 6 native bit error rate. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 402–404. IEEE, 2019.

[221] Hussein Nili, Gina C Adam, Brian Hoskins, Mirko Prezioso, Jeeson Kim, M Reza Mahmoodi, Farnood Merrikh Bayat, Omid Kavehei, and Dmitri B Strukov. Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors. *Nature Electronics*, 1(3):197–202, 2018.

[222] Jayita Das, Kevin Scott, and Sanjukta Bhanja. Mram puf: Using geometric and resistive variations in mram cells. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(1):1–15, 2016.

[223] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[224] Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. An attention free transformer. *arXiv preprint arXiv:2105.14103*, 2021.

[225] Telmo Pessoa Pires, António V Lopes, Yannick Assogba, and Hendra Setiawan. One wide feedforward is all you need. *arXiv preprint arXiv:2309.01826*, 2023.

[226] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE, 2019.

[227] Haocheng Xi, Changhao Li, Jianfei Chen, and Jun Zhu. Training transformers with 4-bit integers. *arXiv preprint arXiv:2306.11987*, 2023.

[228] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701*, 2020.

[229] Sangmin Yoo, Yongmo Park, Ziyu Wang, Yuting Wu, Saaketh Medepalli, Wesley Thio, and Wei D Lu. Columnar learning networks for multisensory spatiotemporal learning. *Advanced Intelligent Systems*, 4(11):2200179, 2022.

[230] Sangmin Yoo, Yuting Wu, Yongmo Park, and Wei D Lu. Tuning resistive switching behavior by controlling internal ionic dynamics for biorealistic implementation of synaptic plasticity. *Advanced Electronic Materials*, 8(8):2101025, 2022.

[231] Sangmin Yoo, Eric Yeu-Jer Lee, Ziyu Wang, Xinxin Wang, and Wei D Lu. Rn-net: Reservoir nodes-enabled neuromorphic vision sensing network. *arXiv e-prints*, pages arXiv–2303, 2023.

[232] Jason K Eshraghian, Max Ward, Emre O Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 2023.