RELIABILITY AND SECURITY OF COMPUTE-IN-MEMORY BASED DEEP NEURAL NETWORK ACCELERATORS

A Dissertation
Presented to
The Academic Faculty

by

Shanshi Huang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2022

RELIABILITY AND SECURITY OF COMPUTE-IN-MEMORY BASED DEEP NEURAL NETWORK ACCELERATORS

Approved by:

Dr. Shimeng Yu, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Saibal Mukhopadhyay
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Callie Hao
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ada Gavrilovska Habl
School of Computer Science
Georgia Institute of Technology

Date Approved: September 26, 2022

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor Dr. Shimeng Yu for his invaluable advice, unreserved support, and meticulous guidance throughout this whole journey. He is the most intelligent, rigorous and hardworking person I have ever met in my life. He has set an excellent example for me regarding what a good researcher (and person) should be. I believe what I have learned from he will benefit me for life. Besides, this endeavor would not have been possible without the financial support of the Semiconductor Research Corporation and National Science Foundation.

I am also very grateful to the rest of my dissertation and proposal committee: Dr. Callie Hao, Dr. Saibal Mukhopadhyay, Dr. Ada Gavrilovska Habl, Dr. Tushar Krishna, Dr. Sung Kyu Lim, for their insightful comments and encouragement. My sincere thanks also go to Dr. Meng-fan Chang and Dr. Jae-sun Seo for their recommendations and our exciting collaborations.

I thank my former and current labmates for the kind help and wonderful time that we experienced together: Dr. Xiaoyu Sun, Dr. Xiaochen Peng, Dr. Panni Wang, Dr. Wonbo Shim, Dr. Jae Hur, Dr. Sola Woo, Yandong Luo, Wantong Li, Anni Lu, Yuan-Chun Luo, Gihun Choe, Chinsung Park, Po-Kai Hsu, Janak Sharda, Jungyoun Kwak, James Read, Vaidehi Garg, Omkar Phadke, Junmo Lee.

Last but not the least, to my parents Jin and Qin, and my husband Hongwu, thank you for supporting and encouraging me throughout this whole journey and my life. I would not have made it without you.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **iii**

LIST OF TABLES **vi**

LIST OF FIGURES **vii**

SUMMARY **xi**

CHAPTER 1. Introduction **1**
1.1 Motivation **2**
1.2 CIM basics **3**
1.3 CIM reliability and security issues **5**
1.3.1 Compute-in-memory reliability issues 5
1.3.2 Compute-in-memory security issues 11
1.4 Thesis overview **13**

CHAPTER 2. Hardware-Aware Quantization/Mapping Strategies for Compute-in-Memory Accelerators **17**
2.1 Motivation **17**
2.2 Low precision neural network **17**
2.3 Neural network mapping **21**
2.3.1 MAC mapping in CIM 22
2.3.2 Analog-to-digital conversion 32
2.3.3 Post-ADC digital processing 34
2.4 Evaluation results **36**
2.4.1 Hardware trade-offs among different mapping strategies 36
2.4.2 Hardware trade-offs with different ADC configurations 40
2.5 Summary **45**

CHAPTER 3. Performance Recovery under Process Variation **46**
3.1 Motivation **46**
3.2 Recover the performance below variation **46**
3.2.1 ADC offset variation modelling 46
3.2.2 On-chip fine-tune 49
3.2.3 Performance recovery 51
3.3 Summary **53**

CHAPTER 4. Achieving High In-Situ Training Accuracy and Energy Efficiency with Analog Non-Volatile Synaptic Devices **55**
4.1 Motivation **55**
4.2 Non-idealities for in-situ training accuracy **56**
4.2.1 Asymmetry/nonlinearity in conductance tuning 57
4.2.2 D2D variation and C2C variation 59
4.2.3 Update step size 60

4.2.4 ADC quantization 60
4.3 Training on chip 61
4.3.1 Training-on-chip architectures 61
4.3.2 Segmented calculation 63
4.4 Evaluation and discussion 65
4.4.1 Software evaluation 65
4.4.2 Hardware evaluation 69
4.5 Summary 74

CHAPTER 5. Secure XOR-CIM Engine: Compute-in-Memory SRAM Architecture with Embedded XOR Encryption 76
5.1 Motivation 76
5.2 Secure XOR-CIM inference engine 76
5.2.1 Secure inference engine system 76
5.2.2 Secure XOR-CIM inference engine architecture 80
5.3 Hardware implementation of XOR-CIM core 81
5.3.1 6T Dual-WL SRAM bit-cell 81
5.3.2 Decrypting input generator 82
5.3.3 ADC pair resolution 83
5.3.4 Dynamic reference array 84
5.4 Evaluation and discussion 84
5.4.1 Methodology and setup 84
5.4.2 Hardware performance of the XOR-CIM core 85
5.4.3 Overhead for secure communication 86
5.4.4 Vulnerability of XOR key bit sharing 88
5.5 Conclusion 90

CHAPTER 6. Secure eNVM-CIM Engine: Exploiting Process Variations to Protect Machine Learning Inference Engine from Chip Cloning and Adversarial Attack 91
6.1 Motivation 91
6.2 Vulnerability of eNVM-based CIM chip 91
6.2.1 Chip cloning attack 91
6.2.2 Adversarial attack 93
6.3 Security benefit 94
6.3.1 Protect machine learning inference engine from chip cloning 94
6.3.2 Mitigating transferability of adversarial examples 96
6.4 Summary 100

CHAPTER 7. Conclusion 101
7.1 Key contribution 101
7.2 Future work 103

REFERENCES 105

LIST OF TABLES

Table 1 - Average percentage of weights updated per iteration. 53

Table 2 - Hardware overhead for fine-tuning of one epoch. 53

Table 3 - Survey of representative analog synapses reported in the literature with weight precision, P/D, and C2C variations. 60

Table 4 - Hardware performance benchmark (Training). 73

Table 5 - Benchmark hardware performance among Normal-CIM, near-memory compute, and XOR-CIM. 86

Table 6 - Hardware performance of on-chip blocks at 28nm for implementing the entire ResNet-18 secure inference engine. 87

Table 7 - Accuracy performance under C&W attack ($\boldsymbol{L_2}$). 98

Table 8 - C&W attack on VGG-8 with different distance matrices. 99

LIST OF FIGURES

Figure 1 - CIM basics. (a) A basic mixed-signal CIM array for VMM acceleration (b) Mapping from a convolutional layer to the CIM subarrays. 4

Figure 2 - Thesis overview 14

Figure 3 - Accuracy performance vs. network quantization approaches with (a) 8-bit weight/8-bit input and (b) 2-bit weight/8bit input. 21

Figure 4 - Subarray structures for different mapping (data representation) methods: (a) Case1: 2’s complement extended representation (b) Case2: differential-pair data representation (c) Case3: shifted unsigned INT. 27

Figure 5 - Input distribution of different quantization methods on (a) VGG-8 network and (b) ResNet-18 network. 28

Figure 6 - Hardware mapping with input-dependent cancellation for (a) Case2: differential-pair data representation; (b) Case3: shifted unsigned INT; (c) Case1: 2’s complement extended representation for binary digit. 31

Figure 7 - Accuracy performance vs. on/off ratio with (a) Case1: 2’s complement extended representation (b) Case2: differential-pair data representation (c) Case3: shifted unsigned INT. 32

Figure 8 - Hardware performance (energy efficiency, throughput, area overhead) vs. different design options (quantization methods, mapping methods, cell precision) with no ADC quantization loss. 38

Figure 9 - Accuracy performance vs. ADC precision for different design options (quantization methods, mapping methods, cell precision). 41

Figure 10 - Hardware performance (energy efficiency, throughput, area overhead) vs. different design options (quantization methods, mapping methods, cell precision) with tolerable ADC quantization loss. 42

Figure 11 - Accuracy performance vs. ADC precision with nonlinear quantization for different design options (quantization methods, mapping methods, cell precision). 44

Figure 12 - (a) Latch-based current-mode SA. (b) Sense pass rate for 5-bit ADC. 47

Figure 13 - (a) Sense pass rate to Iref offset conversion. (b) Sigma/mu of the Gaussian distribution of Iref offset converted from sense pass rate. 48

Figure 14 - Simulated ADC output with offset sampled from the Iref distribution. 49

Figure 15 - On-chip fine-tuning dataflow. 50

Figure 16 - Accuracy before fine-tuning for (a) Flash-ADC and (b) SAR-ADC. 52

Figure 17 - Retraining curve of (a) Flash-ADC with 2-bit weights; (b) SAR-ADC with 2-bit weights; (c) SAR-ADC with 4-bit weights; (d) SAR-ADC with 8-bit weights. 52

Figure 18 - Non-idealities of analog synaptic devices in (b) eNVM array for in-memory computing with (a) asymmetric and non-linear conductance tuning, device-to-device (D2D) variation, and cycle-to-cycle (C2C) variation and (c) ADC quantization error. A reference column by subtraction is used to represent negative weights. 56

Figure 19 - (a) Conductance update trend without momentum. (b) Conductance update trend with momentum. (c) Weight distribution without asymmetry/nonlinearity. (d) Weight distribution with P/D = +3/-3 for Conv layer 3 for the VGG-8 network at epoch=185. 57

Figure 20 - Three types of undesired ΔW sign change during training. 58

Figure 21 - ADC quantization example. 61

Figure 22 - (a) On-chip training architecture with naïve step-by-step dataflow (case1). (b) On-chip training architecture with segmented on-chip accumulation and updates (case2). 63

Figure 23 - Example of input and error feature map reload for case2 segmented gradient calculation. 64

Figure 24 - The number of DRAM access bits for each layer under different global buffer sizes from 16 kB to 1024 kB for case1 (naïve) and case2 (segmented gradient calculation) schemes. 65

Figure 25 - (a) Accuracy vs. momentum factor β. (b) Accuracy vs. device nonlinearity factor under asymmetry. (c) Training traces w/wo momentum, D2D variation and big batch size=4000. 66

Figure 26 - (a) Accuracy vs. momentum precision. (b) Accuracy vs. device weight (gradient) precision. (c) Accuracy vs. C2C variation. Momentum β =0.9 is applied. 67

Figure 27 - Statistical weight update pattern (ΔW vs. W) in one batch for different P/D, C2C variance, and weight (gradient) precision. (a) When C2C variation is small for devices with small P/D and high precision. (b) When C2C variation is large for devices with large P/D and high precision and case (c) When C2C variation is large for devices with small P/D and low precision. 68

Figure 28 - (a) Training traces for different ADC resolutions. (b) Training traces when asymmetry/nonlinearity, D2D variation, C2C variation, momentum, and ADC quantization effects are combined. 69

Figure 29 - (a-c) Energy breakdown for case1 (naïve) dataflow. (b-f) Energy breakdown for case2 (segmented calculation) dataflow. 70

Figure 30 - Area breakdown for (a) case1 (naïve) dataflow (b) case2 (segmented calculation) dataflow. 71

Figure 31 - Energy efficiency vs. global buffer size for case1, case2, and hybrid scheme. 72

Figure 32 - Energy breakdown for hybrid dataflow with the portion of momentum in the weight update highlighted. 73

Figure 33 - Energy consumption & energy efficiency vs. training batch size. 74

Figure 34 - The two-party system of the CIM inference engine and the cloud database, and the possible threats of model leaking on the system. 77

Figure 35 - Communication protocol for the secure SRAM-based CIM inference engine. 79

Figure 36 - Secure XOR-CIM inference engine architecture. 80

Figure 37 - (a) Dual-WL 6T SRAM cell. (b) Decrypting input signal generator. 83

Figure 38 - Inference accuracy vs. ADC resolution. 83

Figure 39 - Latency, energy and area breakdown for the secure inference engine. 87

Figure 40 - Inference accuracy vs. portion of network encrypted. 88

Figure 41 - Illustration of weight encryption for 4-bit weight. 89

Figure 42 - The hamming distance between the predefined key and the key found by frequency analysis with several weights sharing one key. 89

Figure 43 - Chip-cloning attack that bypass the expensive process of data/label collection and model training. 92

Figure 44 - The treat of generating adversarial examples from one chip and affecting all the rest chips. 94

Figure 45 - Inference accuracy distribution of software-trained model with ADC offset. 96

Figure 46 - Three adversarial attack scenarios: Case1: attack the original model in software and apply the examples on a fine-tuned chip; Case2: attack the digital model read out from a fine-tuned chip and apply the generated examples on a fine-tuned chip; Case3: attack the model on-chip directly in a hybrid way and apply the generated examples on another fine-tuned chip. 96

SUMMARY

The proposed research aims to explore the reliability and security issues in compute-in-memory (CIM) design for accelerating deep neural network (DNN) algorithms. On one side, this research focuses on investigating and overcoming the impact of non-idealities in CIM designs. We first explore the design space of the CIM inference accelerator's quantization and mapping strategies. Several typical design options are analyzed and compared from both the software and hardware performance sides. Some design options are more robust and hardware friendly than others, inspiring further improvement in quantization and mapping strategies. The first work considers non-ideal effects from quantization and mapping strategies, with ideal circuits and devices assumed. Considering a more real-life situation, reliability issues caused by non-ideal circuits are studied. Specifically, the process variation is introduced to ADCs of the CIM inference engine, which causes the ADC offset. The effect of ADC offset on the software performance is evaluated, and an on-chip fine-tuning solution is proposed to compensate for the performance degradation. Embracing the benefit of on-chip fine-tuning, we explore the possibility of directly training on-chip of CIM accelerators with analog synapses under the non-idealities of devices and circuits. The in-situ training is proven feasible even under asymmetry/nonlinearity, device-to-device (D2D) variation, cycle-to-cycle (C2C) variation, and a limited number of states.

On the other side, security vulnerabilities and countermeasures for SRAM-based CIM and eNVM-based CIM inference engines are investigated. The SRAM-based inference engine must download the model each time after power-on as it is volatile. Thus,

we propose an XOR-CIM-based inference engine working in a two-party system, in which encryption and authentication are adopted considering data transmission between servers and edge devices. The eNVM-based engines mainly suffer from the information leaking problem brought by raw data stored in non-volatile memory. Inspired by the necessary on-chip fine-tuning to recover the accuracy loss brought by the process variation, a physical unclonable function (PUF)-like scheme is proposed against the weight cloning attack and to mitigate the transferability of the adversarial examples.

CHAPTER 1.   Introduction

Although the concept of machine learning (ML) was proposed a long time ago, its development was lagging due to the limited hardware computing power until graphics processing units (GPUs) were introduced to this area [1]. In recent years, remarkable breakthroughs in machine learning have encouraged applications in various fields, such as computer vision, autonomous vehicles, and natural language processing. These successes in software, in turn, have motivated the development of hardware accelerators from the cloud to the edge. As machine learning models become more and more powerful, their implementations also tend to be more and more data-intensive. Thus, the frequent data movement becomes the bottleneck of the accelerators based on the conventional von Neumann platforms (e.g., CPUs/GPUs). As an alternative solution, compute-in-memory (CIM) attracts more and more attention since it merges the compute units directly into memory units, alleviating the memory wall problem.

The principle of CIM for machine learning acceleration is that the crossbar structure of the memory array could effectively support dot-product operations, which take an extensive part of the calculations in most machine learning applications. Various CIM architectures have been proposed to support different kinds of tasks [2, 3, 4, 5]. This work mainly focuses on the CIM acceleration for the currently most popular machine learning technique, the deep convolutional neural network (DNN). Also, among different types of CIM, this work focuses on the mixed-signal CIM approach, which first processes vector-matrix multiplications (VMMs) in the analog domain and then digitizes the outputs at the edge of the array for further processing.

1.1 Motivation

The concept of edge computing is becoming increasingly popular in this era of big data and machine learning. On one side, it could reduce bandwidth usage and latency in communication. On the other side, it allows applications on edge to be optimized for a particular environment timely.

The most well-known application adopting edge computing is Internet-of-Things (IoT), where the calculations can be done locally in edge smart devices such as smartphones, smart watches, smart home appliances and so on. Powered by machine learning techniques, IoT could mimic intelligent behavior with less human intervention. While IoT does not necessarily require edge computing, processing data locally will greatly improve calculation efficiency and reduce data transfer. It also reduces the risk of personal information leakage, as many smart devices could reflect the owner's behavior and privacy. Another application that embraces edge computing is wearable medical devices, which care a lot for real-time response and privacy. Machine learning applications have been demonstrated to be efficient for various medical purposes. With edge computing, these applications could be personalized to everyone's health condition for better performance. Besides user specialization, edge computing is also beneficial for environmental adaptation. For example, the traffic signals of an intersection could be optimized based on the movements, time series, and environmental variables. For cloud computing, the data from different locations need to be transformed to the centralized server, and then the optimized traffic flow is sent back, introducing big data transfer and delayed response. Thus, edge computing is also suitable in this scenario. Like most planning applications, traffic signals planning could also be powered by machine learning algorithms.

CIM is a promising solution for accelerating these DNN-based machine learning applications at the edge, considering its power efficiency. However, it behaves differently from traditional digital circuits, requiring more cross-layer designs from algorithm levels to hardware implementations. As CIM essentially adopts mixed-signal computations, non-ideal effects are naturally introduced, which could hamper the accuracy. Thus, performance cannot be guaranteed when applications developed in pure digital processors are directly mapped to CIM processors. Thus, how to mitigate non-ideal effects and improve the reliability of CIM engines to maintain the software performance are remaining problems.

On the other side, edge devices suffer from higher security risks than the data center considering their usage environment. While treats are the same whether the edge devices are CIM or digital based, the same solution may not be suitable for both. Due to the weight stationary and parallelism computing nature of the CIM, it needs to be taken care of differently to avoid hurting its efficiency. Generally, the on-chip DNN model, trained with extensive resources, is identified as a valuable asset to be protected. Effective and lightweight countermeasures are desirable to prevent DNN model leaking and reverse engineering.

1.2 CIM basics

The CIM structure discussed throughout this research is based on the mixed-signal calculation unit demonstrated in Figure 1 (a). It utilizes the crossbar structure of the memory array to accelerate vector-to-matrix multiplications (VMMs) in the analog domain. More precisely, the components of a matrix (weights) will be mapped to the memory cells in the array. The scalars in the vector (inputs) will be applied to the cells through the horizontal wires of the array, which are generally the word lines (WLs) of the array. An analog signal will be generated at each cross-point to multiply the input and weight. Generally, the analog signal can be a current generated based on Kirchoff's law. The analog signals from the same column will be accumulated through the horizontal wires, which are usually the array's bit lines (BLs). This way, the array's output will be analog signals proportional to the VMM results.

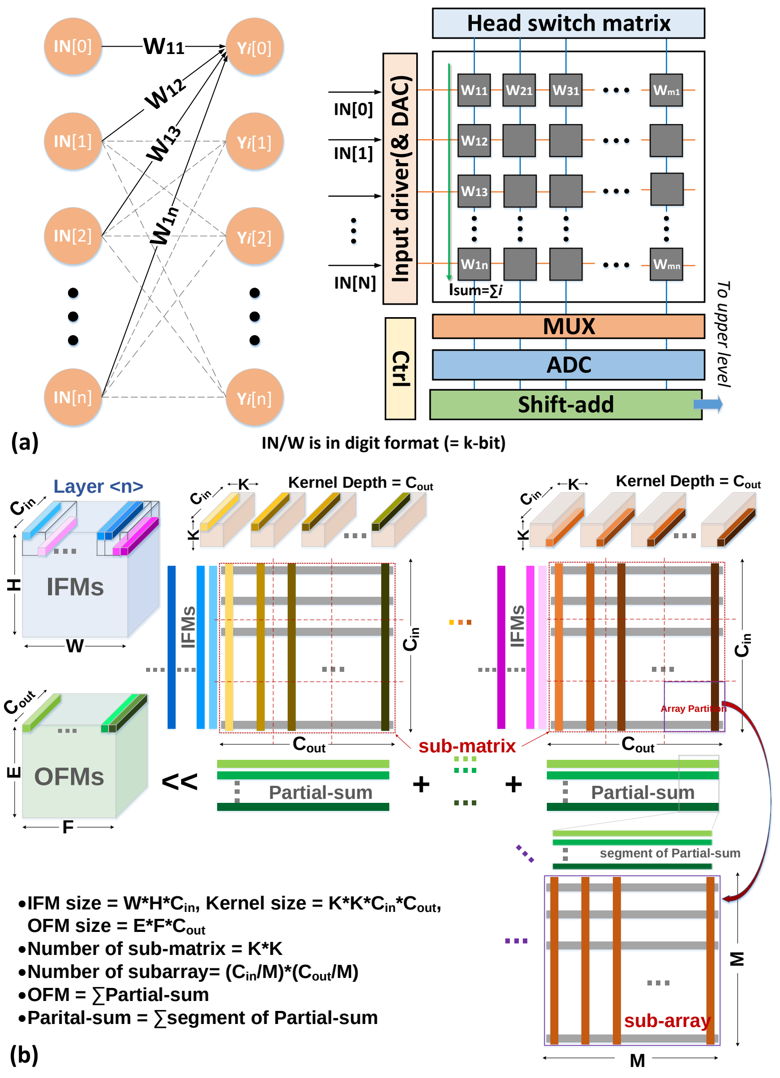


Figure 1 - CIM basics. (a) A basic mixed-signal CIM array for VMM acceleration (b) Mapping from a convolutional layer to the CIM subarrays.

The CIM structure could accelerate the DNN-based applications, which usually contain a lot of VMM operations in the Convolutional (CONV) layers and Fully-Connected (FC) Layers. The operations in FC layers are VMMs in nature. However, the CONV layers usually have to stretch the 4D kernel to a 2D matrix. Considering a weight kernel of a CONV layer has the size $k_1 \times k_2 \times C_{in} \times C_{out}$, the operation of the CONV layer could be viewed as VMM between input vectors and a 2D weight matrix with size $(k_1 \times k_2 \times C_{in}) \times C_{out}$ in principle. In general, this stretched 2D matrix could be much bigger than the CIM array size $(M \times M)$ and thus has to be cut into multiple subarrays. Different stretching/cutting methods have been proposed based on the architecture and data flow. In this research, we adopt the method demonstrated in Figure 1 (b), which views the weight array as $k_1 \times k_2$ 2D sub-matrixs with size $C_{in} \times C_{out}$ first. A sub-matrix could be further divided into multiple subarrays. In this way, there will be in total $k_1 \times k_2 \times (C_{in}/M) \times (C_{out}/M)$ subarrays per weight digit needed for the layer.

1.3 CIM reliability and security issues

1.3.1 Compute-in-memory reliability issues

Many CIM architectures have been proposed based on various memory types. Generally, static-random-access-memory (SRAM) and emerging non-volatile memories (eNVMs) are popular for in-memory computing. On one side, eNVMs are alluring due to their non-volatility, high density, low leakage, and multilevel programmability. On the other hand, the SRAM is highlighted with the fast write speed, low write energy, scalability to more advanced tech nodes, and flexibility of function integration as it is CMOS-based. SRAM-based and eNVMs-based CIM accelerators could suffer from common or peculiar limitations in mixed-signal computation, such as ADC quantization loss, IR drop, read disturbance, process variations, etc.

ADCs will introduce non-idealities into the CIM accelerators regardless of the device used. First, it is usually impractical to use full-precision ADCs in the CIM array considering their size and energy consumption. Reducing the ADC precision will introduce quantization errors, which may diminish the performance of the DNN on-chip. Techniques have been explored to reduce the required ADC precisions while maintaining performance. The ISSAC [6] design reduces the ADC from full precision by 1-bit using weight encoding. The encoding method first checks whether the weights are collectively large for each column of weights. If not, the weights are stored in their original form. Otherwise, the weights are saved in a "flipped" form. In such a method, even with the maximal inputs, the sum of products always yields an MSB of 0, and the 1-bit lower precision ADCs would not introduce quantization error. Later, the AEPE [7] design proposes further reducing the ADCs' precision at a balanced expense of accuracy degradation. By checking the statistics of the DNN value, they find that the most significant bits (MSBs) carry much less information (which means most of the MSBs are small or even zeros) than the least significant bits (LSBs). Thus, a bounding method that only keeps the least significant bits was used to reduce the ADC precision. Evaluation results show that this bounding method introduces negligible accuracy loss with up to 2-bit quantization loss for big networks such as AlexNet [8], VGG-16 [9], and ResNet-50 [10] for ImageNet classification. In [11], Sun et al. utilize non-linear quantization to minimize the ADC precision with negligible accuracy. The non-linear levels are determined via the Lloyd-Max algorithm and mapped by a look-up table. It is shown that for MLP on MNIST classification, the ADC precision can be reduced by 3-5 bits according to the subarray size. For the CNN on CIFAR-10, the acceptable ADC loss is within the range of 2 to 4 bits for different subarray sizes. The prior works are mainly targeted at inference accelerators. For the training engines, the weight statistics vary with time, especially at the beginning, requiring a relatively higher ADC quantization precision.

Another concern caused by digitization is the input offset on analog readout circuits (i.e., sense amplifier) caused by process variations. This offset bias the ADC's references away from their desired value, thus introducing ADC quantization errors. To overcome this problem, Yin et al. [12] compensate for this offset by fine-tuning the references after fabrication. An automatic algorithm is proposed to generate the correction step proportional to the difference between the ideal and actual ADC outputs, scaled by a factor decreasing with correcting iterations. One drawback of this method is that, while the offset is static after fabrication for certain sense amplifiers (SAs), they are random across different SAs. Thus, the references need to be fine-tuned for each SA on-chip. Advanced sensing offset cancellation techniques are proposed with increased circuit complexity as a replacement. For example, a triple-margin small-offset current-mode sense amplifier (TMCSA) is proposed in [13], and a dual-bit small-offset current-mode sense amplifier (DbSO-CSA) is proposed in [14]. Compared to the traditional CSA design, which compares input's and reference's current in two symmetric branches, these advanced designs sense the current difference in the same branch: the input current is copied to the pull-up/pull-down circuit while the reference current is copied to the pull-down/pull-up circuit. In this case, the offset between the input pairs of two branches is avoided. In addition, the TMCSA could amplify the current difference by three times to tolerant small read/sense margins, while the DbSO-CSA uses two reference currents to generate a 2bit sense result, reducing the number of references and sense time.

Wire resistance is also a common problem in all CIM accelerators and will be more and more severe as technode scales down. The wire resistance will cause IR drop, making currents contributed by the cells vary across different cross-points. Liu et al. propose compensation methods for IR-drop in [15]. For the inference engine, they fine-tune the resistance of the cell to make the combination of cell and wire resistance close to the ideal value to represent the weight. For the inference engine, they fine-tune the resistance of the cell to make the combination of cell and wire resistance close to the ideal value to represent the weight. For the training compensation, the programming width is modulated by the cell location, considering the IR drop of the writing voltage. Still, the resistance of the cell is targeted at a value that makes the combination of cell and wire resistance represent the weight. Instead, He et al. treat the IR drop as noise and utilize the noise robustness of the neural network to eliminate its effect [16]. They approximate the IR drop as an additive Gaussian noise at the edge of the crossbar arrays and train the network with noise injected. Tested on a LetNet-5 on MNIST classification, the network still converges with noise injected and shows much better IR-drop robustness.

Besides, some non-idealities are caused by circuit structures and thus vary from one design to another. One drawback is the nonlinearity in the analog output signal caused by the readout circuits. For example, if resistors or capacitors are used as loads of current mirrors or convert the current to voltage, the readout currents/voltages may saturate as partial sums increase. One straightforward solution is to map ADCs' references accordingly, considering the nonlinearity, as shown in [17]. Instead, Yoon et al. [18]

propose a readout circuit with input-aware current control and a feedback amplifier to force linear output voltage at the expense of increased energy consumption.

Read disturbances exist in both SRAM-based and eNVM-based CIMs but work differently. For the 6T-SRAMs, the VMM is implemented by activating several rows simultaneously. If the bit line (BL) voltage drops below the write margin due to numerous pull-down branches, the nodes storing “1” will be flipped. One straightforward way to avoid data flipping is to limit the voltage swing on BL, decreasing the voltage difference between different partial sums. A more practical way is to use SRAM cells with the decoupled read port, as in many previous works [19] [20] [21]. The read disturbance of eNVMs defines the phenomenon that the read current gradually shifts the cell’s conductance. As presented in [22], a higher read current will cause a larger shift, indicating that a small read voltage is preferred. However, as a drawback, the sense margin of the ADCs will be limited. Chen et al. [23] point out that the single-level cell is preferred over the multilevel cell in terms of robustness to read-disturb. The work in [22] also observes that the middle levels have a relatively weak disturbing immunity.

Another issue in 6T SRAM is the distinct behaviors between input “1” multiplied by weight “0” and input “0” multiplied by weight “1”. When the input is “1”, it is represented as the on-state of the access transistor. Thus, if the weight stored in the Q node is “0”, there will be a discharge path contributed by this “0-1” combination. On the contrary, if the input is “0”, the access transistor will be off, and no discharge/charge path will exist. This asymmetry will cause input-dependent analog output shift and make it hard to design ADC references. Similar to the read disturbance problem, this issue could be solved by the decoupled read port for product operation.

Compared to SRAM, eNVM suffers from more device non-idealities. Some non-idealities matter only in training, while others degrade the inference performance. First of all, the eNVM cells have limited dynamic ranges, which defines the on/off ratio between the maximum conductance (G_{max}) and the minimum conductance (G_{min}). As the cells use different conductance levels to encode weights, the dynamic range will limit the number of levels the cell can represent. For example, STT-MRAM usually has a small on/off ratio and can only be used as binary cells, while ReRAM can be used as multi-level cells [24]. Another issue caused by the small dynamic range is the small difference between the currents contributed by G_{max} and G_{min}. Normally, the weight “0” is represented by G_{min} and 0V represents input "0". In this case, input “0” will contribute no current, while the G_{min} with non-zero input will still cause some current. If the on/off ratio is big, the current contributed by non-zero input and weight “0” can be neglected. Otherwise, the G_{min} will cause the same input-dependent analog output shift problem as the 6T SRAM. In [25], a dynamic reference generation scheme was used to solve this problem. Instead, Luo et al. [26] find that the current contributed by G_{min} could be substracted and has a very limited impact on the performance if a reference column is used. Furthermore, while a single dynamic range is reported as the average case for a type of device, different cells of the same device could have conductance variation. Luckily, this variation is tolerable in two aspects. On one side, the neural network itself is noise-robust to some extent. On the other side, this variation can be mitigated by the write-verify scheme to write the cell [27]. Apart from the dynamic range, data retention is also a concern that limits the number of levels of the cell.

The eNVM cells can be viewed as analog synapses in training. The number of levels defines the smallest step that can be taken to update the cell value. The dynamic range still matters, and a high on/off ratio is desired. On the contrary, retention is no longer a problem since the cell is updated frequently. For most eNVMs, the trajectory for potentiation and degradation is non-linear and asymmetrical [28]. As reported by [29], the non-linear but symmetrical update of weight conductance will not cause a big accuracy loss for training. At the same time, the non-linearity combined with asymmetry will greatly degrade the training performance. From the hardware side, people solve this by introducing capacitors to hold part of the weights during training as it is much more linear, such as PCM+3-transistor-1-capacitor [30] or 2-transistor-1-FeFET [31]. From the software side, a tiki-taka algorithm is proposed for the training with non-linearity/asymmetry [32]. Moreover, each cycle's conductance change could be different even for the same writing pulses, introducing noise in weight update. Finally, the endurance of the cell will also limit the training performance.

1.3.2 Compute-in-memory security issues

Besides the reliability issues related to the CIM architecture, various security concerns different from the traditional von Neuman architecture are also introduced. As mentioned before, the CIM architecture takes advantage of the crossbar structure to accelerate the VMM operation with weight stored at the cross point, which forces the weights stored in memory to stay in their raw format. However, using CIM on edge devices makes it unsafe to directly save the data in raw format, especially for eNVM-based CIM.

In [33], the authors propose a defense method to thwart the Replication Attack against the memristor-based neuromorphic computing system (MNCS). Their system assumes the drone with the ability of on-chip training instead of inference only. The drone is initially untrained and needs to request the training datasets from the base station. Before the drone is used for inference, a secure session will be established under some authentication protocols. An encrypted training set will be sent to the drone, where it will be decrypted for training. The well-trained model on-chip will fail to work after N times of inferences due to the read disturbance. Then, the drone must require the training set again to recover the performance. The authors claim that the MNCS could defend against eavesdropping and spoofing attacks by utilizing authentication and encryption protocols. The probing attack is also considered impossible because of the high density of memristors and compact 3D stack structure. The only concern is the chosen input attack, which uses the drone to generate custom input/output pairs to infer the weights. Once the network is well-trained for the memristor-based system, anyone with physical access to the drone could generate input/output pairs using it. These pairs could be used to infer the model one-chip. To defend against this kind of attack, they designed their MNCS such that the drone would work well for N times. After that, the accuracy will degrade rapidly and need to be recovered with the training set from the base station. If an adversary obtains a drone with a well-trained model, he/she cannot pass the authentication protocol to get the set. Thus, he/she could get no more than N pairs of data, which are not enough for the weight inference.

On the contrary, another work [34] still treats the probing attack as a threat and proposes a framework consisting of sparse fast gradient encryption (SFGE) method and runtime encryption scheduling (RES) scheme to defend against it. The model stored on-chip is encrypted with a small offset added to some of the weights in the proposed framework. The key of SFGE is composed of the encrypted location and the encrypted sign. The sign is found through the gradient so that the performance of the model on-chip could be killed by just tuning a small number of weights with a small value. The encryption of SFGE is just the addition of the weights and the key, while the decryption is the subtraction. There will be a trade-off between the overhead and the encryption effectiveness for this SFGE method. The paper shows that different networks might require a different number of weights per layer to be encrypted to achieve desired performance degradation. Overall, the bottom line of the requirement is small compared to the whole network and thus makes the overhead acceptable. During the calculation, the weight is decrypted to the raw format for CIM operation. Thus, the RES is used to avoid the adversary interrupting the system during runtime and getting an unprotected model on-chip. Thanks to DNNs' layer-by-layer nature, the decryption/encryption of the weights could also be done in this way. Thus, only one layer is in plaintext during runtime and will be immediately encrypted after work. The RES will hide the decryption of the next layer and the encryption of the previous layer under the operation of the current layer. Thus, the latency overhead is small.

1.4 Thesis overview

This thesis addresses the reliability and security issues related to CIM accelerators. The reliability issues caused by non-idealities from architecture, circuits, and devices are discussed with improvement methods proposed. Security vulnerabilities are considered based on the device type used for the CIM accelerator. Lightweight countermeasures are proposed correspondingly. The thesis is organized as shown in Figure 2.

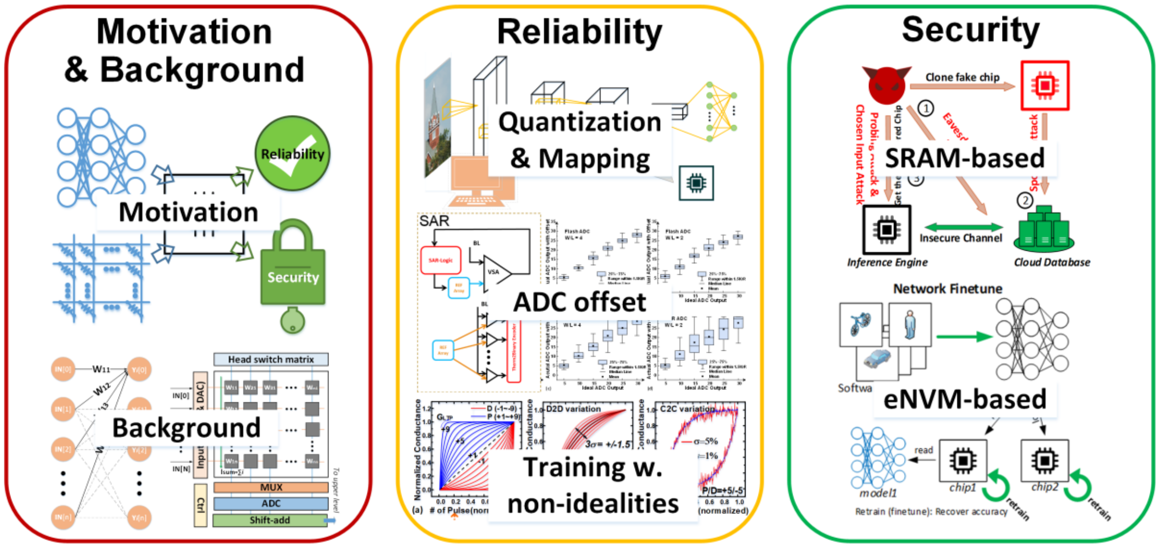


Figure 2 - Thesis overview

Chapter 1 gives an overview of the background of the thesis, including the motivation for studying the reliability and security issues in CIM accelerators, the basics of CIM for DNN accelerations, and the state-of-the-art works related to the reliability and security of CIM.

Chapter 2 discusses the design flow and typical options for mapping the DNNs to the CIM architectures [35]. The reliability and hardware performance of combinations of two quantization methods (i.e., DF and WAGE) and three number mapping schemes (2's complement, differential pair, and shifted unsigned INT) are evaluated. Different ADC schemes (linear vs. nonlinear) are also studied for better implementation. The evaluation results show that the energy efficiency could be improved by ~2× with 1.2~1.6× throughput and 5%~25% smaller area by optimizing the design options of DNNs mapping only.

Different reliability under small on/off ratios and ADC quantization are also shown for different design options.

Chapter 3 introduces the effect of process variation on the CIM accelerator, which mainly causes the sense amplifier (SA) mismatch and, thus, the analog-to-digital converter (ADC) offset [36]. A model of generating ADC offset from the sense pass rate tested from SA is proposed. With the proposed model, the software performance of CIM accelerators with different ADC topologies (SAR-ADC and Flash-ADC) is evaluated. Flash-ADC shows better robustness under process variation. Meanwhile, a hybrid fine-tuning scheme is proposed to compensate for the accuracy degradation caused by ADC offset.

Chapter 4 explores the potential of on-chip training with analog-synapse-based CIM [37, 38]. The non-idealities like nonlinearity/asymmetry of potentiation and depression, device-to-device variation, cycle-to-cycle variations, and ADC quantization for both inference and backpropagation are studied for training. The momentum solution with stochastic gradient quantization is proposed to overcome the drawbacks of software performance caused by the non-idealities. In addition, a segmented gradient calculation method is proposed to reduce the DRAM access in training to maintain the hardware performance.

Chapter 5 presents a lightweight SRAM-based CIM inference engine with a protocol for chip authentication and key processing [39, 40]. An XOR-CIM core is demonstrated with Dual-WL 6T SRAM cells, which integrate the XOR decryption into the CIM operation. By utilizing the XOR-CIM core, the model in transmission and stored on-chip could always be encrypted. The overhead from protocol and protection is proven small on energy, latency, and area by utilizing partial encryption.

Chapter 6 presents the vulnerabilities and countermeasures specific to eNVM-based CIM accelerators [36, 41]. The chip-cloning attack and transferability of adversarial examples are demonstrated as threats to the eNVM-based CIM chip for edge devices. The on-chip fine-tuning to recover the performance degradation under process variation are proved efficient in defending these two treats with very small hardware overhead.

Finally, Chapter 7 summarizes the contribution of this thesis. Future work is also proposed in this chapter.

CHAPTER 2. Hardware-Aware Quantization/Mapping Strategies for Compute-In-Memory Accelerators

2.1 Motivation

Various CIM designs implemented with different kinds of memory cells have been recently proposed for edge inference of CNN [6, 7, 42], and several macro chips have been demonstrated with impressive energy efficiency [11, 12, 43]. While there is a demanding interest in CIM accelerators, studies mostly focus on hardware development on the macro level. However, due to the analog processing manner in the CIM crossbar structure, mapping a DNN model from a digit system to CIM macros is not straightforward. Thus, neural network mapping (NNM) strategies are needed to fill the gap between them. In previous CIM works, each design applies variations of the mapping strategies based on the designers' intuitions, which cover merely a small portion of NNM's large design space. There is no clear reason why some options are chosen over others, making it hard for the following designers to do a comprehensive early-stage NNM design instead of random attempts. However, mapping methods could introduce different quantization losses or reliability under non-ideal effects, leading to different hardware and software performances for the same task. It is important to understand the reasons causing these differences for better mapping strategies designed for good hardware performance and reliability.

2.2 Low precision neural network

Today’s AI researchers tend to improve the DNN’s software performance with larger and larger networks without considering the hardware cost, making them originally unfriendly for edge devices with limited area and power budgets. To solve this problem, people propose compact network structures [44, 45] or quantization on large networks [46, 47, 48, 49] to reduce the computation and memory costs. The compact networks are unsuitable for CIM-based inference engines since they still adopt floating-point computations that are not friendly to the crossbar structure of the CIM array. Moreover, they usually contain depth-wise convolution layers, which have small fan-in and thus are not efficient to be implemented with CIM arrays. On the contrary, quantized networks are hardware friendly to CIM accelerators as they usually assume fixed-point VMM operations with low precision parameters.

We can view the network quantization as a process to map the high-precision floating-point inputs/weights to the low-precision fixed-point inputs/weights without hurting the models' software performance. Previous works [47, 48, 49, 46, 50, 51] have proved that DNNs could be quantized to 1~8 bit weights/inputs with negligible accuracy loss. Theoretically, the hardware performance could be improved by decreasing the VMM parameters' precision. Thus, most network quantization algorithms are aimed at minimizing the parameters' precision. However, the scalability of aggressive quantization methods to large networks for complicated tasks could be a problem. Normally, the network quantization approaches could be sorted into post-training quantization or quantization-aware training. On average, the latter category has presented a more aggressive precision reduction, with some even adopting binary input/weight [50, 51]. Considering the generality of a practical design from the hardware's perspective, less aggressive quantization-aware training methods with good scalability and flexibility will be preferred for the CIM inference engine [47, 48, 49, 46]. These works usually directly use the parameters' precision to evaluate the efficiency of hardware performance improvement without real evaluations done on the hardware side. There is no clue whether quantization methods can make a difference in real implementation under the same parameter precision. In this work, we compared two typical quantization-aware training algorithms and discussed their properties from the hardware's perspective.

$$Y = C \cdot (X + D) \tag{1}$$

Linear quantization could be viewed as an affine map from the real value (Y) to the quantized value (X) as equation 1 in a general format [47]. X generally represents a sub-range of Y since it has lower precision than Y, which we call a quantization range. The quantization range selection is critical to the software performance of the quantized networks. From previous work, there will not be a big change in the statistics of the DNN's weights during training, and quantizing them into a range between $[-1,1]$ will not cause a significant loss in the software performance. Compared to the weights, inputs could vary a lot during training and across different network structures and normally require careful quantization range selection with some preprocesses for quantization.

We group the quantization methods into two categories based on whether the quantization range is flexible. The first one uses a fixed range across all the layers, which could be viewed as a fixed-point quantization. This method is possible since there are normally some normalization layers before the CONV/FC layers in DNNs, so the CONV/FC layers will see similar statistics during training and across layers. A typical example is the WAGE method [46], which always clips the input range to $[-1,1]$ with some factors for normalization between layers. Since these normalization factors are precalculated before training and kept fixed during training, the scalability of WAGE is proven poor as the network goes deep or the network structure becomes complicated. The upgraded WAGE [52] introduces batch-normalization (BN) (i.e., WAGEBN) to replace the fixed normalization factor while still keeping the $[-1,1]$ input quantization range. As BN will calculate the normalization factor on the fly, the new version becomes more robust across different networks. However, BN only works on the mean and standard deviation of the inputs to format the statistics instead of fitting the output into the quantization range we picked. Thus, when passing the BN output to the CONV/FC layer, there might still be some overflow or underflow loss that hurts the software performance.

On the contrary, the other category tries to dynamically decide the quantization range from the input statistics during training. In [48], this quantization with a dynamically decided range is called a dynamic fixed-point (DF) number scheme. This DF could be explained by equation 1 as different input (Y) range are mapped to the same fixed-point number set X by using flexible C. This method requires extra calculation to decide the dynamic ranges during training. Once the training is done, the ranges are fixed during inference but could differ from layer to layer. As the range is directly calculated from the inputs, it will have less probability of causing a big overflow or underflow loss. There are different ways to decide the dynamic ranges from the inputs. In this work, we assume the simplest one, which takes the maximum value of the absolute inputs.

This work compared the hardware performance of these two quantization methods, namely WAGE and DF, as illustrations to show that the quantization method will affect the data distribution and further change the hardware performance even though the same precision is adopted. Figure 3 shows the software-trained accuracy results of these two quantization methods on the VGG-8 network for CIFAR-10 classification and the ResNet-18 network for CIFAR-100/ImageNet (subset) classification. Both quantization methods could achieve the floating-point baseline accuracy with 8-bit input and 2 to 8-bit weight.

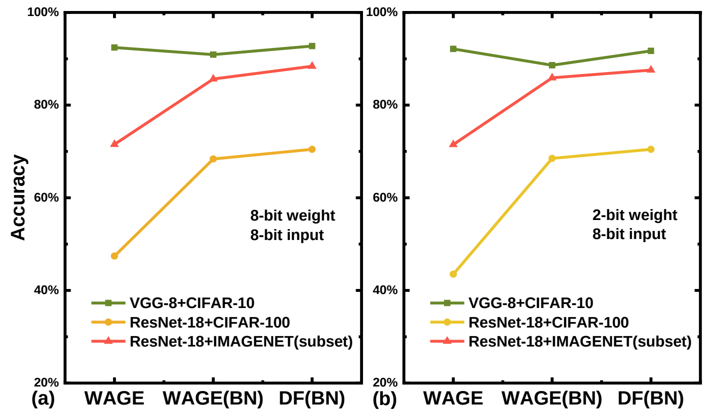


Figure 3 - Accuracy performance vs. network quantization approaches with (a) 8-bit weight/8-bit input and (b) 2-bit weight/8bit input.

2.3 Neural network mapping

After quantizing the high-precision floating-point parameters (Y) of the CONV/FC layers into low-precision fixed-point values (X), it will be much easier to map them to the CIM macro. We divide the mapping of DNN to the CIM architectures into three steps. Firstly, as most of the CIM architectures for DNN adopt a mixed-signal scheme that conducts MAC operations in the analog domain, we need to decide how to map the inputs and weights of MAC operations into the analog representations. Then, as some inter-layer operations, such as BN, ReLU, and pooling functions, are still done in the digital domain, we need to define the process of converting the analog MAC results back to digital signals, namely, the ADC scheme. Finally, for a functional DNN engine, we must define the components used to do the digital processing. Although the hardware implementations could vary from design to design, their principles are common. In this work, we summarize some typical options from the previous designs and show their trade-offs.

2.3.1 MAC mapping in CIM

2.3.1.1 Number system in CIM.

We treat the mapping from MACs to the CIM array as a three-component decision: the number system, the number representation, and the analog mapping method. While digital computer systems are typically binary number systems, the CIM architecture could support high-precision digits in the MAC operation due to analog calculation. In other words, inputs or weights for MAC operation in CIM could be two or more bits per digit. Here, we use digits to denote the units of inputs/weights for one operation in real hardware implementation. An 8-bit number requires eight digits if a binary unit is assumed, while four digits with 2-bit units. Input digits could be multi-bits by equipping input-encoding circuits such as DACs, while weight digits could be increased by adopting high-precision memory cells. For a fixed input/weight precision, higher precision input digits will require fewer computation cycles, and higher precision weight digits could reduce the needed memory capacity for a model. While increasing the digits' precision looks beneficial for both inputs and weights, the precision of analog MAC outputs will also be increased. As the circuit's dynamic range is usually limited, increasing the MAC output precision will reduce the noise margin and increase the ADC resolution and overhead. Theoretically, increasing the digits' precision for weights (cells) or inputs has the same impact on increasing the full precision of MAC outputs. However, it is more expensive to increase the input digit precision considering the encoding circuits than utilizing high-precision memory cells for high-precision weight digits from a hardware resources point-of-view. Generally, reducing the hardware cost is more critical than speeding up the computation for the resource-constrained edge devices. Thus, the number system we employ for the CIM architectures discussed in this work will be binary inputs with flexible precision weights.

2.3.1.2 Number representation in CIM.

As both WAGE and DF utilize a symmetric quantization range around zero, we use N-bit integers (INT) as X (in equation 1) to keep consistency and for easy representation in hardware. Different zero-centered input or weight quantization ranges of Y could be mapped to INT with different scalar values C. In this way, the CIM arrays always see INT operators for MAC operation. Then, we must decide how to encode these INT numbers in the CIM system. The first scheme that comes to mind would be the two's complement representation, as it is the most widely used one in the traditional digital computer for integer representation and calculation. The two's complement representation can naturally encode the sign in the binary sequence for signed calculation. However, sign extension is required for operands in multiplication, making it inefficient for CIM architecture. Instead, since CIM adopts digit-decomposed MAC operations, we could utilize the weighted sum representation of the two's complement data for calculation, as shown in equation 2. This way, all the digits in MAC operations are unsigned, and the signed bases will be introduced after the MAC of digits is done. In more detail, considering the multiplication of input and weight in binary format as equation 2, the unsigned bit-wise multiplication (AND) results b_n of different rows will be accumulated first as a bit-wise partial sum. Then, these bit-wise partial sums will be scaled by their bases 2^n, which could be simply realized by a shift operation in the binary domain. The scaled results could be directly accumulated to the final partial sum if they have positive bases. Otherwise, an additive inverse operation must be applied first.

$$x = b_{N-1} \cdot (-2^{N-1}) + \sum_{n=0}^{N-2} b_n \cdot 2^n \tag{2}$$

While 2’s complement representation is binary, this weighted sum format could be extended to arbitrary-precision digits by grouping k bits together, resulting in high-precision digits with the power of 2^k bases. As an illustration, a 2-bit digit representation is shown below in equation 3. Two adjacent binary bits b_{2n+1}, b_{2n} could be combined as one 2-bit digit $d_n \in [0,1,2,3]$ with a new base $(2^2)^n$. However, this combination is only true when the bases of bits have the same sign. Thus, for the most significant bit b_{N-1}, it could not be grouped with its neighbor b_{N-2}. In this case, b_{N-1} and b_{N-2} could still be viewed as 2-bit digits to keep consistency with the rest digits but only utilize part of the number range. Any signed INT could be represented by the 2-bit digits as equation 4, with even N assumed to illustrate the ungroupable MSB.

$$b_{2n+1} \cdot 2^{2n+1} + b_{2n} \cdot 2^{2n} = (b_{2n+1} b_{2n}) \cdot 2^{2n} = d_n (2^2)^n \tag{3}$$

$$x = b_{N-1} \cdot (-2^{N-1}) + b_{N-2} \cdot (2^{N-2}) + \sum_{n=0}^{\frac{N-2}{2}-1} d_n (2^2)^n \tag{4}$$

For easy reference later, we call this 2’s complement extended representation method Case1, which has been adopted in previous CIM designs [53, 54]. A more general format for this representation is shown in equation 5. Considering an N-bit number represented by k-bit digits in this format, $\lceil (N-1)/k \rceil + 1$ terms are needed in total. Here, the number of terms could be viewed as the number of cycles needed to represent full precision inputs or the number of cells needed for full precision weights. The array structure needed to support this representation is shown in Figure 4 (a).

$$x = b_{N-1} \cdot (-2^{N-1}) + \sum_{n=0}^{\left\lceil \frac{N-1}{k} \right\rceil - 1} (d_n) \cdot\ (2^k)^n,\ d_n \in [0,1,2,\dots,2^{k-1}] \tag{5}$$

Since the sign bit could not be grouped with other bits for high-precision digit extension of the two's complement representation, we could avoid it by grouping the positive and negative weights separately and using the operator to represent the sign information. In other words, the signed number could be represented by a differential pair of two unsigned numbers at the digit level, as shown in equation 6, referred to as Case2. The d_n^+ will represent the digits from a positive weight with the corresponding d_n^- to be zero, while a negative weight will have d_n^- to represent its absolute value with zero d_n^+. The number of terms will become $\lceil (N-1)/k \rceil$ in this case. However, since each term denotes a pair, the number of cycles or cells needed will be doubled. When k is small, corresponding to N, this method may cause a big hardware overhead. A basic array structure of Case2 is shown in Figure 4 (b), and this differential-pair data representation is frequently used in CIM designs [42, 55].

$$x = \sum_{n=0}^{\left\lceil \frac{N-1}{k} \right\rceil - 1} (d_n^+ - d_n^-) \cdot\ (2^k)^n,\ \ d_n^+, d_n^-\ \in [0,1,2,\dots,2^{k-1}] \tag{6}$$

An alternative solution to avoid sign bit is to avoid negative values of X, which utilize the shift D (shown in equation 1) to map Y to unsigned INT as X. The digit-level representation of the unsigned INT is shown in equation 7. The shifted unsigned representation, referred to as Case3, requires $\lceil N/k \rceil$ terms for the high-precision extension, which is the smallest among these three methods for k>1. However, it will cause an extra MAC computation because of the shift D. Considering an example of mapping the weights to the unsigned INT, a positive shift D_w is used to map the weight W^r to X_w^r (equation 8), where the superscript r denotes the weight from different rows. Then, the MAC operation between the weights (Case3) and inputs (Case1) will become equation 9, which is the MAC of unsigned weights and signed inputs with an additional MAC term between the inputs and the shift. If inputs are also represented in Case3, more additional MACs will be introduced. Figure 4 (c) shows an array structure for Case3 that utilizes a dummy column of D_w to generate term $\sum_r D_w \times IN^r$. The subtraction could be done in the digital domain. Ref. [6] has adopted this scheme for the CIM array design.

$$x = \sum_{n=0}^{\lceil \frac{N}{k} \rceil - 1} d_n \cdot (2^k)^n \tag{7}$$

$$X_w^r = W^r + D_w \tag{8}$$

$$\sum_r (X_w^r - D_w) \times IN^r = \sum_r X_w^r \times IN^r - \sum_r D_w \times IN^r \tag{9}$$

As discussed in 2.3.1.1, we utilize binary inputs with flexible precision weight digits in this paper, considering the hardware limitation. For simple hardware implementation, we always use two’s complement representation for input since it is naturally compatible with binary data in the digital system. In other words, inputs always utilize the Case1 mapping strategy while Case1 to Case3 will be evaluated for weight mapping.

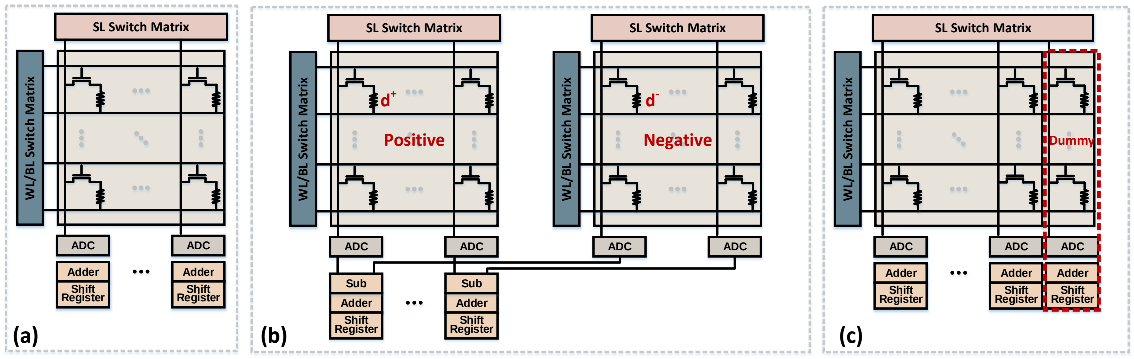


Figure 4 - Subarray structures for different mapping (data representation) methods: (a) Case1: 2's complement extended representation (b) Case2: differential-pair data representation (c) Case3: shifted unsigned INT.

2.3.1.3 Hardware implementation

To realize the operation in circuits, we still need to decide how the real signal encodes the parameter digits. As we utilize binary inputs to avoid extra encoding circuits, the "0" and "1" could be represented separately by 0V and a read voltage Vread. In this way, the CIM structure could automatically skip the zero digit in the input since the input "0" will not contribute current to the partial sum and thus consume no energy. Thus, a higher portion of "0" in the inputs means less energy consumed during calculation. In other words, high sparsity in inputs is preferred. The sparsity of the input will be directly affected by the quantization method due to the quantization range, as shown in Figure 5. Compared to DF, which dynamically calculates the quantization range, WAGE utilizes a hard-clipped range and thus has wider spread inputs with fewer "0" input digits. Thus, even though different quantization methods can achieve similar software performance with the same parameter precision, they can still have different real hardware performances due to input statistics.

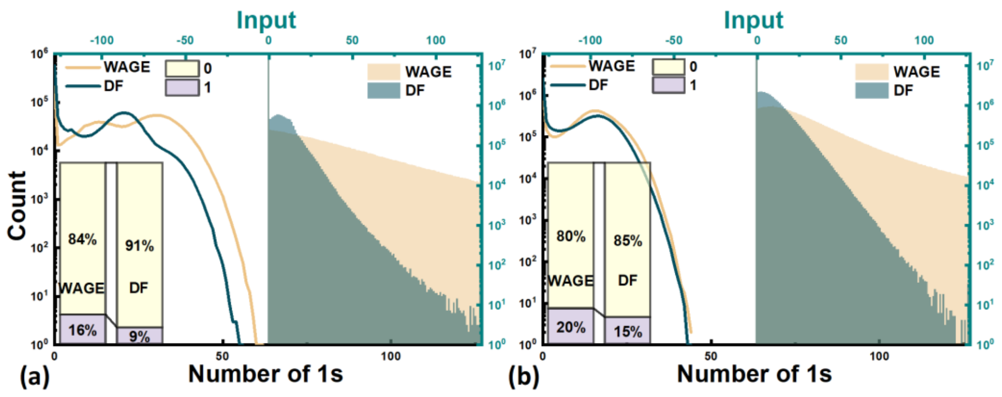


Figure 5 - Input distribution of different quantization methods on (a) VGG-8 network and (b) ResNet-18 network.

Conversely, weight digits are usually encoded to different cell conductance. Generally, small weight digits will be mapped to low conductance values, contributing smaller currents than large conductance cells under input “1”. Thus, the sparsity of the weights is also preferred for high energy efficiency. In this work, we hard-clip the weights into [-1,1] for both WAGE and DF as we do not see much performance or distribution difference across quantization schemes. As a result, the difference in hardware performance between quantization methods is mainly caused by the input statistics. Other techniques, such as pruning [56] or training with a regularizer [57], may be adopted to improve energy efficiency from the weight side.

While the quantization methods affect inputs more than weights, the mapping methods matter much to weights. Equation 10 shows a general format of mapping a k-bit weight digit d_n to a certain conductance value G_n of a k-bit eNVM cell.

$$G_n = d_n \times \Delta G + G_{min}\, , d_n \in [0, 2^k - 1],\ \ \Delta G = \frac{G_{max} - G_{min}}{2^k - 1} \tag{10}$$

In this format, the conductance of the eNVM cells is divided into 2^k levels between G_{max} and G_{min}, and the minimum weight digit “0” is mapped to a non-zero conductance value G_{min}. Thus, mapping from digits to the cell conductance is a shifted scaling with a shift G_{min}. By integrating this representation into the digit-wise analog MAC operation of the i^{th} digit of input and n^{th} digit of weight across different rows (r), we could see that the first term is proportional to the MAC results in the digits domain with a second input-dependent output shift term generated in the real circuit implementation, as shown in equation 11.

$$I_{i,n} = \sum_r V_i^r \left(d_n{}^r \times \Delta G + G_{min}\right) = \sum_r V_i^r \left(d_n{}^r \times \Delta G\right) + \sum_r V_i^r \, G_{min} \tag{11}$$

This input-dependent output shift will make the summed current ($I_{i,n}$) for the same ideal partial sum vary with input patterns, which could cause difficulty in ADC design. The variation contributed by the shift term may be negligible when the on/off ratio of the cell is high with low cell precision, in which case the ΔG is much larger than G_{min}. However, for cells with a low on/off ratio or high precision, the current caused by this shift term could be comparable with the main part, causing overlaps among the analog output of different partial sums. As a result, a fixed reference ADC will introduce quantization errors when converting the analog signal back to the digital domain, harming the accuracy performance. The flexible reference ADC could solve this problem with additional hardware [39]. Conversely, this input-dependent output shift could be canceled according to the mapping method with no penalty.

Mathematically, Case2 and Case3 mapping methods could digit-wisely cancel G_{min} in nature. As shown in equation 12, the shift term will be canceled between the positive and negative digits for the differential pairs (Case2).

$$\begin{aligned} I_{in} &= \sum_r V_i^r \left({d_n}^{r+} \times \Delta G + G_{min}\right) - \sum_r V_i^r \left({d_n}^{r-} \times \Delta G + G_{min}\right) \\ &= \sum_r V_i^r \left({d_n}^{r+} \times \Delta G\right) - \sum_r V_i^r \left({d_n}^{r-} \times \Delta G\right) \end{aligned} \tag{12}$$

Similarly, the input-dependent output shift term could also be canceled by the dummy column D in the Case3 approach, as shown in equation 13.

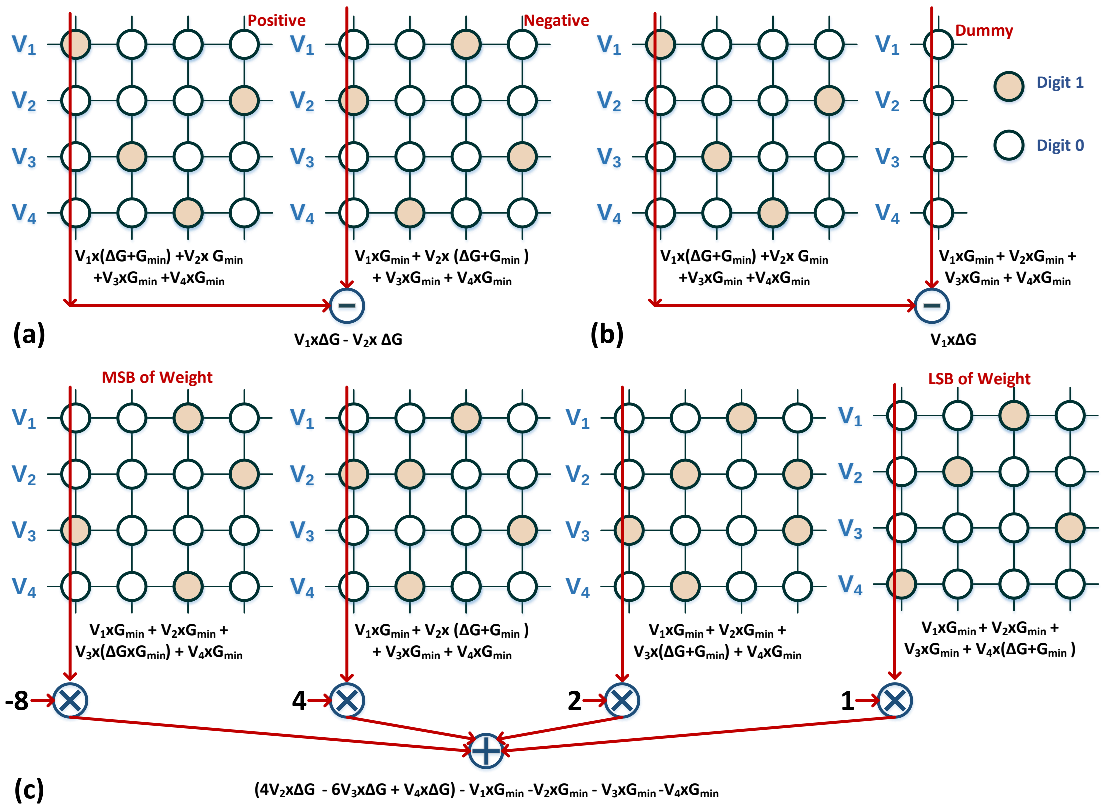
$$\begin{aligned} I_{ij} &= \sum_r V_i^r \left({d_n}^{r} \times \Delta G + G_{min}\right) - \sum_r V_i^r \left({d_{D_n}}^{r} \times \Delta G + G_{min}\right) \\ &= \textstyle\sum_r V_i^r \left({d_n}^{r} \times \Delta G\ \right) - \sum_r V_i^r \left({d\_D_n}^{r} \times \Delta G\ \right) \end{aligned} \tag{13}$$

There is no digit-wise cancellation in the Case1 implementation. However, this shifted term will be diminished by the shift-add operation across weight digits with different bases, as shown in equation 14. Considering the input digit (V_i) applied to the binary weights digits from the same full precision weights, a common term ($\sum_r V_i^r\ G_{min}$) will be generated with different bases. By grouping the bases first, we could see that the full precision shift will only be a small term compared to the full precision MAC result since the bases will cancel each other. However, as the digit precision increases, the cancellation among bases will decrease. In this case, inspired by Case3, we could use a dummy column of all-G_{min} cells to cancel the input-dependent output shift term.

$$I_i = I_{iN-1} \cdot (-2^{N-1}) + \sum\nolimits_{n=0}^{N-2} I_{ij} \cdot 2^j \tag{14}$$

$$I_{i_{shift}} = \sum_r V_i^r\, G_{min} \cdot (-2^{N-1}) + \sum\nolimits_{n=0}^{N-2} \sum_r (V_i^r\, G_{min}) \cdot 2^n$$

$$= \sum\nolimits_r V_i^r\, G_{min} \left(\sum\nolimits_{n=0}^{N-2} \cdot\ 2^n - 2^{N-1}\right) = -\sum\nolimits_r V_i^r\, G_{min}$$



Positive
Negative
Dummy
Digit 1
Digit 0
V₁ V₂ V₃ V₄
V₁x(ΔG+Gmin) +V₂x Gmin +V₃xGmin +V₄xGmin
V₁xGmin + V₂x (ΔG+Gmin) + V₃xGmin + V₄xGmin
V₁xΔG - V₂x ΔG
(a)
V₁x(ΔG+Gmin) +V₂x Gmin +V₃xGmin +V₄xGmin
V₁xGmin + V₂xGmin + V₃xGmin + V₄xGmin
V₁xΔG
(b)
MSB of Weight
LSB of Weight
V₁xGmin + V₂xGmin + V₃x(ΔGxGmin) + V₄xGmin
V₁xGmin + V₂x (ΔG+Gmin) + V₃xGmin + V₄xGmin
V₁xGmin + V₂xGmin + V₃x(ΔG+Gmin) + V₄xGmin
V₁xGmin + V₂xGmin + V₃xGmin + V₄x(ΔG+Gmin)
-8
4
2
1
(4V₂xΔG - 6V₃xΔG + V₄xΔG) - V₁xGmin -V₂xGmin - V₃xGmin -V₄xGmin
(c)

Figure 6 - Hardware mapping with input-dependent cancellation for (a) Case2: differential-pair data representation; (b) Case3: shifted unsigned INT; (c) Case1: 2's complement extended representation for binary digit.

As a more straightforward illustration, we use an example of 4-row accumulation, assuming 4-bit weight with binary digits, to show shift cancellation for different mapping methods in Figure 6. Here, we ignore the impact of ADC quantization on cancellation effectiveness. No mismatch will be introduced to the equations (12-14) shown above if the cancellation is made before ADC, but it is hard to realize the cancellation in the analog domain. While the ADC is considered and subtraction is done in the digital domain, the shift may not be fully canceled because of the ADC quantization error. Proper ADC precision is required to make the remaining shift small in this case.

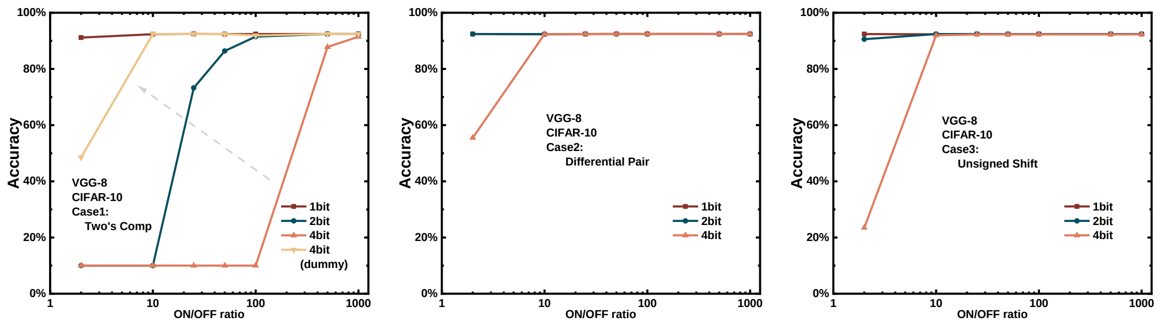


Figure 7 - Accuracy performance vs. on/off ratio with (a) Case1: 2's complement extended representation (b) Case2: differential-pair data representation (c) Case3: shifted unsigned INT.

Figure 7 shows the accuracy performance of the VGG-8 network with 1/2/4-bit weight digits under different cell on/off ratios and number representations methods. As expected, the Case1 method could maintain the accuracy with binary cells until a very small on/off ratio. The robustness disappears as the cell precision increases since the base cancellation no longer holds. For the 4-bit per-cell condition, a more than 1000 on/off ratio is needed for negligible accuracy loss. Luckily, we could release the on/off ratio requirement back to ~10 for the Case1 mapping method with 4-bit cells by utilizing an all-zero dummy column. For Cases2 and Case3, as the digit-wise G_{min} cancellation mechanism will not disappear with increased cell precision. The accuracy could be maintained across different precision settings with a large on/off ratio range, as shown in Figure 7 (b) (c).

2.3.2 Analog-to-digital conversion

The conversion from the analog to the digital domain is the most important part of the mixed-signal CIM and is also the hardware performance bottleneck. As mentioned in 2.3.1.1, the efficiency of the analog MAC could be improved by utilizing high-precision digits, which could either accelerate the computation or reduce the required CIM arrays. However, this will also increase the precision of the partial sum, requiring a high-resolution ADC for lossless conversion. Generally, as the ADC is area-consuming, it is hard to fit one ADC for every column pitch. A common solution is to have several columns share one ADC through a multiplexer at the expense of reduced column-wise parallelism. Since the ADC area is usually proportional to its precision, the higher the ADC precision adopted in the CIM array, the lower the column-wise parallelism. Besides the area, as illustrated in previous work [58], the ADC also dominates the energy consumption and latency in CIM design, which increases with the ADC precision. Finally, due to the limited dynamic range, the noise margin will also decrease with increased ADC precision, introducing more ADC quantization errors and hurting the software performance.

Two strategies could be used in analog output digitization to release this ADC limitation caused by the lossless conversion of a high-precision partial sum. The first one avoids high-precision partial sums by utilizing low-precision digits and reducing the number of rows open in parallel [59]. In this case, only a low-precision ADC is needed for the lossless conversion. This technique is mainly used to relieve the noise margin problem and fits more ADCs into peripheral circuits of a single array, leading to high column-wise parallelism. However, this will neither reduce the energy consumption nor speed up the calculation as the row-wise parallelism is sacrificed. As lossless conversion with a high noise margin is utilized in this method, the software performance of this method could be guaranteed across different structures or tasks. The other method gives up the lossless conversion as DNN is generally sparse and noise-tolerated. Due to the sparsity, the partial sum distribution will be concentrated around part of its representation range, giving us room to reduce the ADC precision. As an example mentioned in 1.3.1, AEPE [7] cut MSBs of the partial sums as they are “0”s in most cases. In addition, because of the noise tolerance, even if the ADC quantization loss changes the partial sum, the final software performance could still be maintained. There is no guarantee of how much quantization loss each network could tolerate, so the reliability issue will be introduced if ADC quantization loss is allowed. In principle, we want to minimize the effect of ADC quantization loss on the software performance with the possible maximum ADC precision reduction. Statistically, the quantization error of the partial sum could be minimized by fitting the ADC quantization to the partial sum distribution. Based on this fact, Sun et al. [11] use the Lloyd-Max algorithm to find nonlinear references for quantization. While this method could reduce the expectation of the partial sum quantization error than linear quantization, a look-up table (LUT) is needed to map the ADC levels to real digital values for further processing. By allowing the quantization loss in the analog to digital conversion, both the noise margin and the parallelism will be maintained with reduced energy consumption and latency from ADC. However, since the acceptable quantization loss is based on its effect on the software performance, we need to explore the best settings for each task and model. Furthermore, since the ADC precision loss is achieved from the partial sum distribution, it could also vary for the same task and model when adopting different quantization and mapping methods.

2.3.3 Post-ADC digital processing

After conversion, ADC outputs will be further processed in the digital domain. Mostly, digital circuits could be fixed-point or floating-point. The fixed-point circuit will be more area and energy efficient than the floating-point unit (FPU). Similar to the CONV/FC layer, other operations of DNNs are originally floating-point in general-purpose processors. Further processes are needed when mapping some functions (e.g., BN) to fixed-point circuits. Thus, for simple implementation, we combine the fixed-point circuits with FPUs to support different functions in this work.

No matter which mapping scheme is used, if the weight/input digit precision is smaller than the parameter precision, the digit-wise MAC results need to be shifted & added by the digital circuit. Moreover, if the weight matrix partition is employed, the partial sums from different subarrays are added using pure digit circuits. These operations could be fixed-point, whose precisions are affected by the ADC design. If the ADC adopts linear quantization references, the ADC output will be used directly as the digital signal. Thus, the ADC precision straightly decides the following digital circuits' precision. However, for nonlinear ADC quantization, the output of LUT will decide the digital signal precision, which is normally higher than the ADC precision. Thus, while the nonlinear ADC shows more aggressive ADC precision reduction, it may cause increased overhead in the following fixed-point digital circuits. The CIM combined with the shift & add and sub-array addition will compromise the CONV/FC layer computation. Thus we could say the whole CONV/FC layer is fixed-point.

According to the network structure or quantization method, some intermediate functions, such as BN and/or scaling, must be applied between two layers of Conv/FC. From previous works [49] [46], it is hard to convert these functions to fixed-point without further processing or adopting high precision. Also, compared to the CONV/FC layers, these functions only take a small part of the computations in DNNs. Since this work mainly focuses on the CIM structure for CONV/FC operation, which dominates the system-level performance, we use a FPU to support these operations. We also assume that if there are back-to-back BN and Scaling in the network, they will be merged into one FPU operation, which could further reduce the hardware consumption of these operations.

2.4 Evaluation results

2.4.1 Hardware trade-offs among different mapping strategies

Hardware performances across different design options are first evaluated with full precision ADC. All the designs are evaluated at the 22nm node in DNN+NeuroSim V1.3 [60] with RRAM CIM arrays. The device' Ron/Roff is assumed to be 6kΩ/900kΩ [61]. For different networks discussed in this work, we choose the array size to be the minimum kernel $input\ channel\ depth \times output\ channel\ depth$ of the network (except for the first layer) for better memory efficiency. Our evaluations have been done on VGG8 for CIFAR10 classification and ResNet18 for CIFAR100 and IMAGENET (partial) classification. Without specification, the parameter precision is 8-bit/8-bit for inputs/weights (8b-W/8b-IN).

In this assumption of the similar CIM macro and lossless ADC, the hardware performance differences are purely caused by quantization or mapping methods. According to Figure 7 (a), a dummy column is necessary for Case1 under 4-bit cell precision under the utilized RRAM on/off ratio. Figure 8 summarizes the trends of the chip area, energy efficiency, and throughput across different design settings. First, under the same design options, we find that DF always reports better energy efficiency than WAGE. This result matches our observation in section 2.3.1.3 that there are more “0”s in the DF’s input bits compared to WAGE, making the former more energy efficient. At the same time, the quantization algorithms make little difference in the area overhead/throughput, which is mainly affected by the hardware resource overhead/dataflow. Compared to WAGE, DF needs one more step of input scaling. However, as the scale is merged with BN and CONV/FC layer computation hardware dominates the chip area/processing time, the difference is almost negligible. Thus, we could conclude that DF quantization is a better choice than WAGE considering better energy efficiency with full-precision ADC.

Figure 8 (a-c) show the chip areas of different networks/tasks under different hardware settings. When 1-bit cells are adopted, the number of terms (in equations 5-7) is the same for all the mapping methods. Case2’s chip area is much bigger than the rest two since it takes two memory arrays to represent one weight digit term. Case3 is slightly bigger than Case1 because of the additional dummy column. As the cell precision increases, Case1 needs one more term than the other two and an extra dummy column for G_{min} cancellation. Thus, the difference between Case1 and Case2 gradually decreases, and Case3 becomes the most area efficient. In conclusion, with full precision ADC, Case2 is dominant in the area overhead. Case1 is more area efficient when cell precision is low, while case3 is better when cell precision is high.

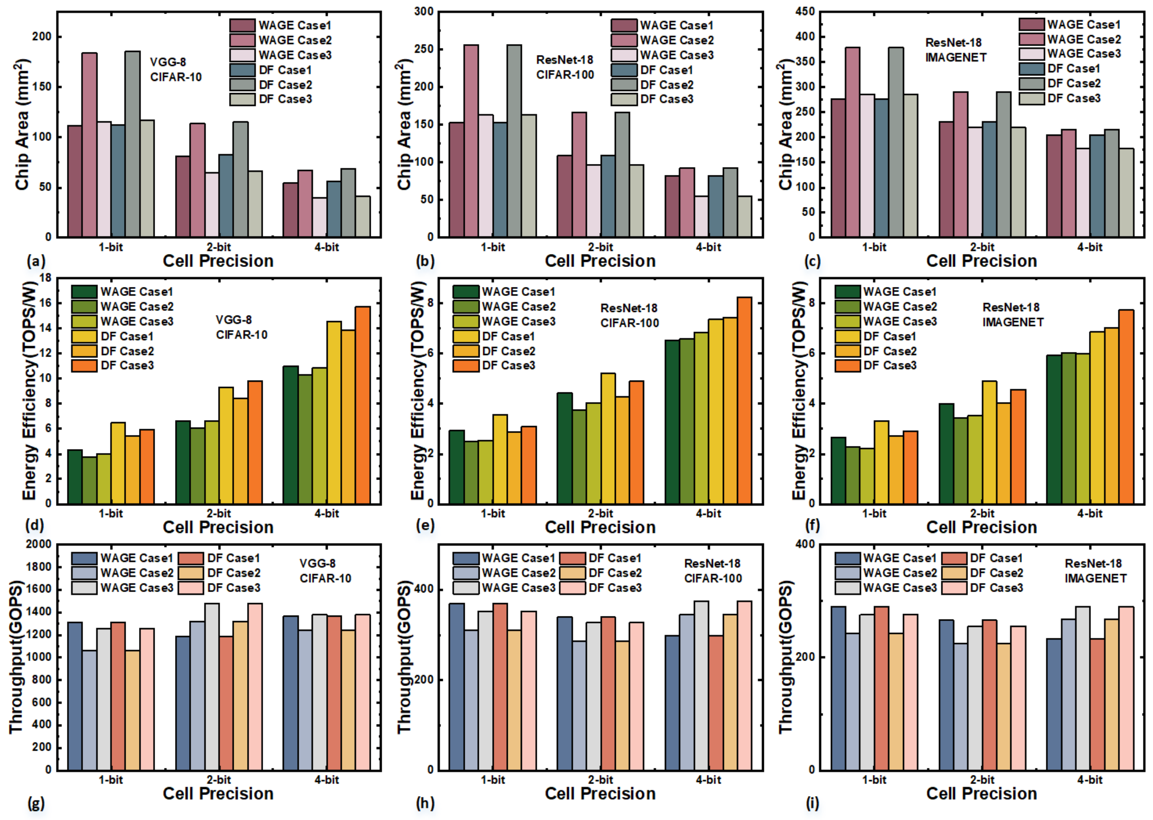


Figure 8 - Hardware performance (energy efficiency, throughput, area overhead) vs. different design options (quantization methods, mapping methods, cell precision) with no ADC quantization loss.

As the CIM is proposed as an energy-efficient edge accelerator, we care more about the energy efficiency results in Figure 8 (d-f). Since extra hardware means more components to consume power, Case1 gives the best energy efficiency for the 1-bit cell precision, with Case2 being the worst. However, as energy efficiency is also affected by the input/weight statistics, the area overhead and energy efficiency trends do not strictly match. When the cell precision goes high, Case1 worsens because of the extra sign bit, while Case3 wins again in energy efficiency.

We also observe that the throughput has no clear trend across cell precisions and mapping methods, as shown in Figure 8 (g-i). By analyzing the latency contribution of each stage, we find that the CIM system's latency mainly consists of the array latency and the network-on-chip (NOC) latency. Unlike the area and energy cost dominated by array computing, the array latency and the NOC latency could be comparable under some settings. On the array side, the latency is highly affected by peripheral circuits such as ADC. If full-precision ADC is considered with the same cell/input precision, the array latency will be almost the same across mapping methods. The array latency will increase when ADC precision increases with the cell precision. NOC's latency is jointly determined by the number of bits to transmit, the wire length, and the interconnection network complexity. The chip area related to mapping methods has already been discussed before. Under the same parameter precision, the higher the cell precision is, the smaller the chip size will be, as fewer weight arrays are required. As a result, the NOC latency should decrease as the wire length and interconnection network complexity tend to be reduced. However, the increased cell precision will also raise the ADC's full precision, leading to wider output bit width. More cycles will be needed for data transfer under the fixed bus width, and the NOC latency will be increased. As a result, the NOC latency has no clear increase or decrease trend with the cell precision, leading to an unclear trend for system throughput.

In conclusion, we could optimize the design options for neural network quantization/mapping without ADC quantization loss based on previous findings. The quantization method giving higher sparsity is always preferred under the same parameter precision. With a certain quantization approach, when the cell precision is low related to the weight precision, Case1 is suggested. Case3 should be considered as the cell precision increases for better energy efficiency and area overhead. The optimal design option for the three evaluated networks/tasks could achieve a 29%~45% improvement in energy efficiency from the worst ones with 4%~40% area reduction and 4%~25% speedup under the same cell precision.

2.4.2 Hardware trade-offs with different ADC configurations

As discussed in 2.3.2, the hardware performance of the CIM accelerator can be improved by allowing quantization loss in the analog-to-digital conversion without hurting the software performance. The quantization reference should fit the partial sum distribution, which the network, task, quantization, and weight mapping methods could impact, to maximize the precision reduction. Thus, there is no straightforward way to decide the proper ADC precision reduction except for testing by simulation from case to case. In this work, we assume the same ADC references set for different layers of the network and different components of the MAC, considering the reference generation overhead. We sweep the ADC precision for different networks/tasks/hardware settings to check the effect on the software accuracy (Figure 9).

There is no consistent trend between WAGE and DF about which one is more ADC quantization robust across different hardware settings. If we assume the minimum accepted accuracy for each task to be 90%, 67%, and 83% for CIFAR-10, CIFAR-100, and ImageNet (subset) classification, the minimum ADC precision required is quite similar between WAGE and DF, which means no more than 1-bit difference. In other words, the quantization methods show different robustness to the ADC quantization loss because of the different parameter statistics. However, this difference is insufficient to cause a significant difference in ADC precision requirements.

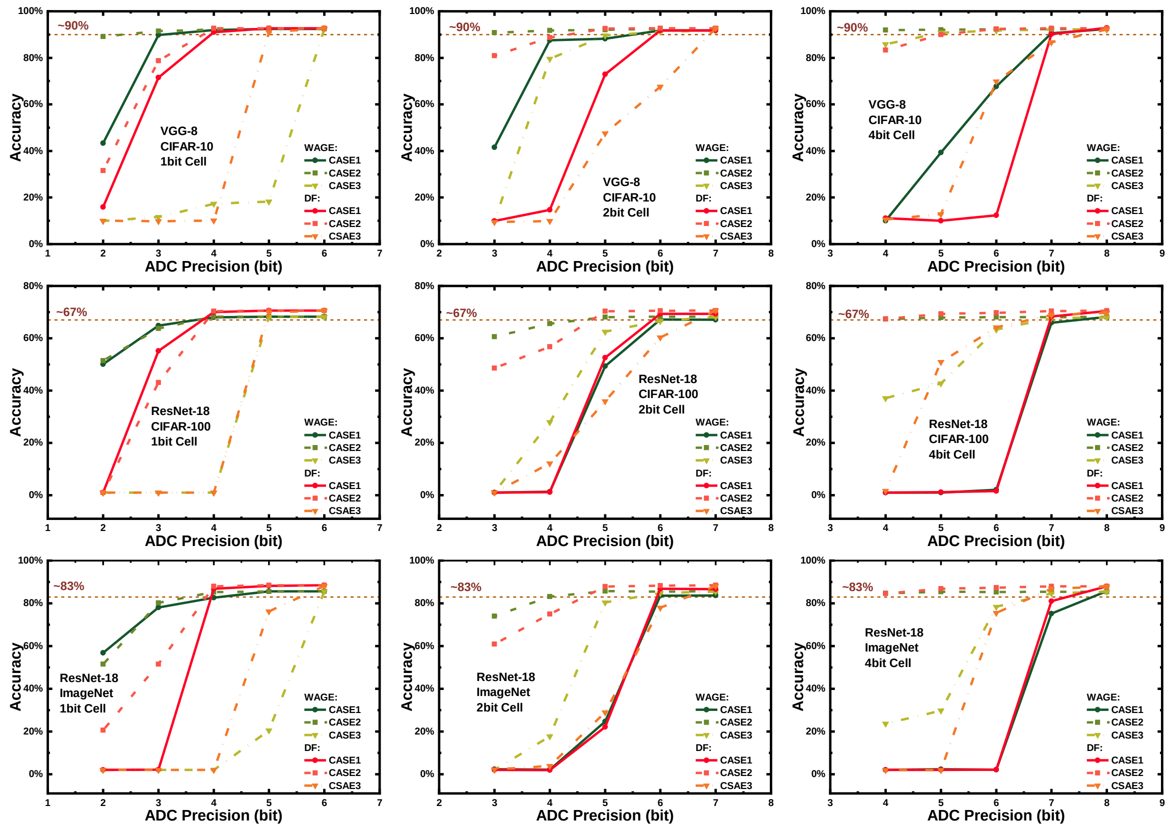


Figure 9 - Accuracy performance vs. ADC precision for different design options (quantization methods, mapping methods, cell precision).

On the contrary, mapping methods affect ADC precision significantly. Under the same quantization method, Case2 is almost the most robust one to ADC quantization loss across different cell precision. Again, Case1 is good when the cell precision is low, while Case3 is better with high cell precision. The big difference among these mapping methods considering ADC quantization is caused by the fact that Case1 and Case3 see two significantly different partial sum distributions from the CIM array. For Case1, the digit encodes the sign bit always contains a binary value. Thus, when the cell precision is high, the partial sum distribution generated from the sign digit and other digits will be very different, requiring a high-precision ADC to cover both cases. Similarly, the partial sum distribution of the dummy column differs from the data columns in Case3, especially when the cell precision is low. Thus, Case1 and Case3 ADC quantization could be improved by applying different ADC for the sign array or dummy column at the expense of additional ADC references.

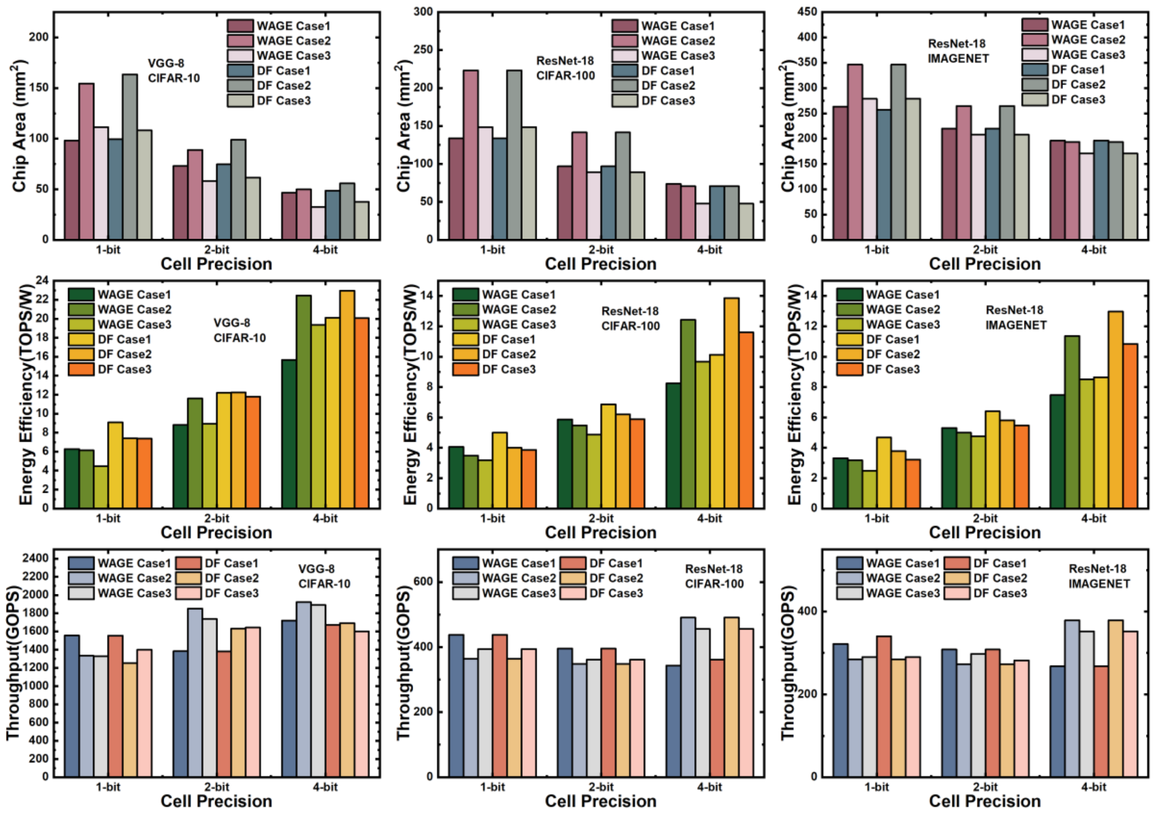


Figure 10 - Hardware performance (energy efficiency, throughput, area overhead) vs. different design options (quantization methods, mapping methods, cell precision) with tolerable ADC quantization loss.

Next, we evaluate the hardware performance with optimized ADC precision. Figure 10 (a-c) show a similar trend with the full precision ADC case on the overhead area. However, since the ADC dominates the area CIM array, the area overhead of Case2 becomes less dominant compared to the other two cases as it could tolerate higher ADC precision reduction. From the energy efficiency result in Figure 10 (d-f), Case1 is still the best choice considering energy efficiency for low-precision cells. As the cell precision increase, Case2 shows better energy efficiency, benefiting from the aggressive ADC precision reduction.

In the 4-bit per cell case, this ADC difference even makes Case2 of WAGE outperform Case1/Case3 of DF, which has higher input sparsity. Thus, we could say that these quantization/mapping design options collectively determine the energy efficiency of the CIM accelerator. Finally, the different ADC precision makes the array latency vary across mapping strategies, making the throughput even harder to predict (Figure 10 (g-i)).

To further reduce the ADC precision to improve the hardware performance, we also test the nonlinear ADC quantization, as mentioned in [11]. In detail, the partial sums from all layers in a network are collected, and the Lloyd-Max algorithm is used to find the nonlinear quantization levels from these partial sums. The accuracy results of different tasks are shown in Figure 11. Compared to the linear quantization (Figure 9), the accuracy of Case1/Case2 methods drops slower with nonlinear quantization. Unanticipated, the nonlinear quantization does not work for Case3 in our test. We think this is because Case3 hires dummy columns to shift the unsigned output back to the signed partial sum. While the dummy columns are important for correct output, their outputs are not statistically dominant in the partial sum distribution. As a result, the nonlinear quantization levels will be ignorant of these dummy statistics, causing big quantization errors. These quantization errors will be global biases and cause big differences in partial sums. Again, a potential solution could be assigning different ADCs for dummy output at the expense of hardware overhead for additional reference.

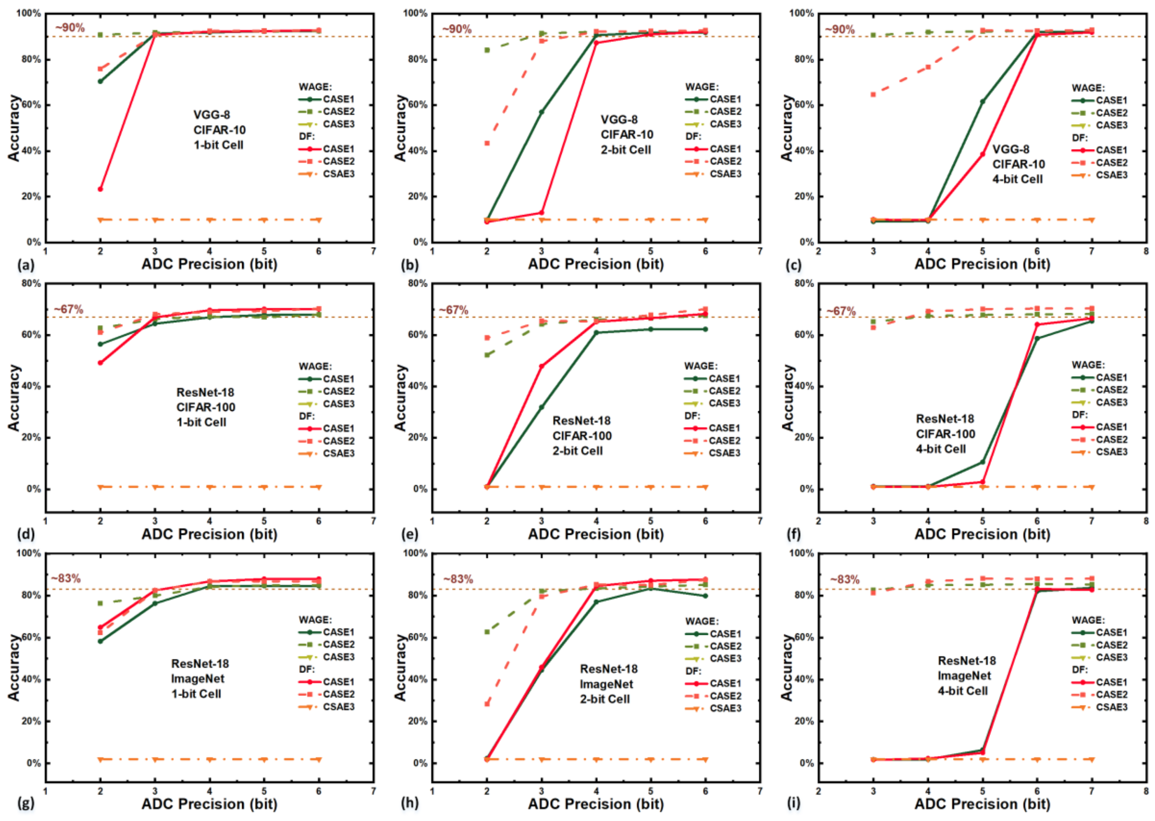


Figure 11 - Accuracy performance vs. ADC precision with nonlinear quantization for different design options (quantization methods, mapping methods, cell precision).

Another unforeseen finding is that there could be an initial loss of nonlinear quantization accuracy when the ADC precision is high for some settings (e.g., the accuracy performance shown in Figure 11 (e) from 6-bit to 4-bit). The problem is that when we exploit the Lloyd-Max algorithm to find the nonlinear quantization levels, we fit them to the distribution of a collected partial sum from different layers. However, when the quantization is applied in a certain layer, it will change the input distributions of the following layers. Thus, the global nonlinear quantization levels are no longer the best fit for the following partial sums. In other words, the software performance of nonlinear quantization by the Lloyd-Max algorithm is not guaranteed for high ADC precision across different networks and tasks, making it an unsafe choice. Also, even for good-performing settings, there is generally no more than 1 bit saving on the ADC precision from linear quantization. Moreover, the nonlinear quantization requires LUT and higher precision fixed-point circuits for the following computation, introducing extra hardware overhead. According to these results and analysis, we can infer that nonlinear ADC quantization could not effectively benefit CIM designs.

2.5 Summary

This work demonstrates a basic flow of the algorithm-to-hardware mapping from DNN to the CIM architecture. The design space of this flow is explored to provide a deeper insight into the system-level CIM design. In detail, we analyze the effect of parameter statistics on the hardware performance by comparing two software quantization methods (i.e., DF and WAGE) with three number mapping schemes (2's complement, differential pair and shifted unsigned INT). We also compare the effectiveness of linear and nonlinear ADC quantization schemes. Our evaluation results show that the quantization/mapping options are very important in determining the statistics of the weight and input digits used in the CIM operations. On one side, these statistics will determine the energy consumed by the array. On the other side, they will determine the partial sum distribution and directly affect the ADC precision needed, leading to different robustness under ADC quantization loss and hardware performance. Also, linear ADC is preferred over nonlinear ADC, considering the hardware overhead, limited precision reduction, and reliability across tasks. Tested on three different tasks, the optimized design options for neural network mapping can improve energy efficiency by ~2× and throughput by 1.2~1.6× while reducing 5%~25% area overhead from the worse cases.

CHAPTER 3. Performance Recovery under Process Variation

3.1 Motivation

In the previous section, we only introduce the effect of quantization and mapping methods on CIM architecture's software and hardware performance. The circuits and devices are assumed ideal in the evaluation, which is not true in real life. As mentioned in 1.3.1, the non-ideal circuits and devices could degrade the software performance of the CIM accelerators. From prior works, one of the non-idealities in circuits is that the process variation can introduce ADC offset and hurt the network performance. In this work, we analyze the ADC offset in circuits and compensate for the software performance loss caused by it using software and hardware co-optimization.

3.2 Recover the performance below variation

3.2.1 ADC offset variation modelling

Generally, two ADC topologies are popular in CIM architectures: Flash-ADC and successive-approximation-register (SAR)-ADC. For an N-bit ADC, a Flash-ADC utilizes $2^N - 1$ comparators to generate a thermometer code, which must be encoded to a binary signal, while the SAR-ADC uses one comparator but N cycles to generate a binary sequence directly. Due to its high sense speed and low power consumption, the sense amplifier (SA) is regularly used as the comparator in ADC. Different kinds of SAs could be used in CIM, which could generally be grouped into the current-mode sense amplifier (CSA) and the voltage-mode sense amplifier (VSA) based on the input signals.

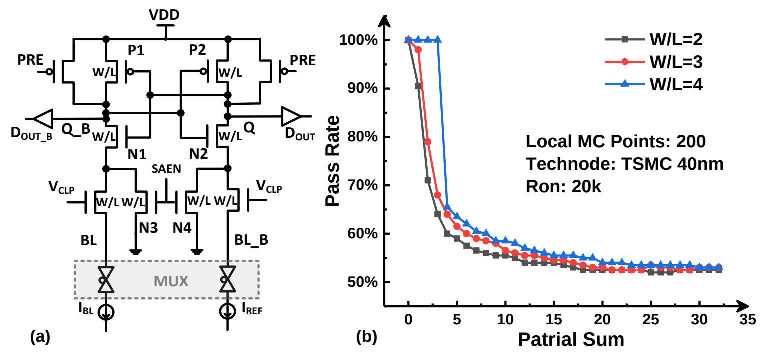


Figure 12 - (a) Latch-based current-mode SA. (b) Sense pass rate for 5-bit ADC.

Figure 12 (a) demonstrates a simple CSA with a small overhead area. In principle, this structure should be symmetric. When the two branches see different pull-down currents, the side with a larger pull-down current will produce the final output “1” while the other will get “0”. However, the mismatch of the two branches caused by process variation can overwhelm the difference between the pull-down currents and cause wrong output, namely ADC error. In a case study of a 5-bit ADC in Figure 12 (b), the sense pass rate, which is defined as the percentage of correctly sensing the I_{BL} comparing to its nearest I_{REF}, will decrease with the increase of the partial sum. A similar trend is reported on the silicon data, and we think two reasons could explain it. One is that as the partial sum increases, the corresponding I_{BL} and its nearest I_{REF} will also increase. There will be a big voltage drop on the transistors of the SA, making the voltage difference caused by the ADC offset dominate. Another one is that, as the I_{BL} increases, it will gradually saturate due to the readout circuit, making the current difference between different partial sums smaller, leaving less room between I_{BL} and I_{REF}.

In this work, a model is proposed for the SA offset caused by the process variation for simulation. We assume that the offset of the SA could be converted to the shift of reference current away from its ideal value. The output will be wrong if the reference shifts to the other side of the partial sum (Psum). Suppose the reference shift follows the Gaussian distribution. In that case, the sense pass rate could be interpreted as the cumulative probability that references smaller than Psum, as shown by the green shaded part in Figure 13 (a). Then, the Gaussian distribution could be uniquely defined by the mean, the ideal value, and the standard deviation, inferred from the sense pass rate.

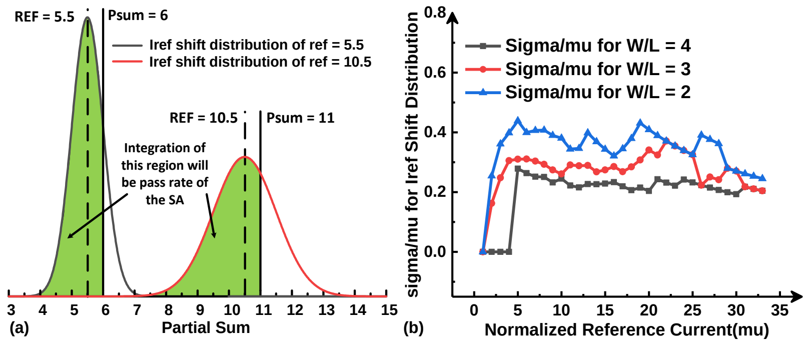


Figure 13 - (a) Sense pass rate to Iref offset conversion. (b) Sigma/mu of the Gaussian distribution of Iref offset converted from sense pass rate.

Since the SA offset is a static offset caused by manufacturing, it will not change with time. For a 5-bit Flash-ADC, as one SA is used for each reference level, different references could shift in different directions by different distances. On the contrary, the SAR-ADC uses the same SA for different levels. Thus, all the references should be shifted to the same side with dependent steps. Based on this fact, if the Flash ADC is assumed in the design, we will sample one offset for each reference from the corresponding Gaussian distribution. As to SAR ADC, one offset is sampled from the Gaussian distribution of a certain reference with non-zero standard deviation, and the offset of the rest levels will be scaled by the sigma/mu of each level, as shown in Figure 13. Figure 14 shows the ideal ADC output vs. ADC output with offset based on the proposed method to integrate process variation into ADC simulation. The Flash-ADC has less ADC error than the SAR-ADC since the independently biased levels could compensate for each other. Here, the thermometer-to-binary encoder is simply an adder tree. Increasing the transistor size will also reduce offset caused by variation and thus reduce ADC error.

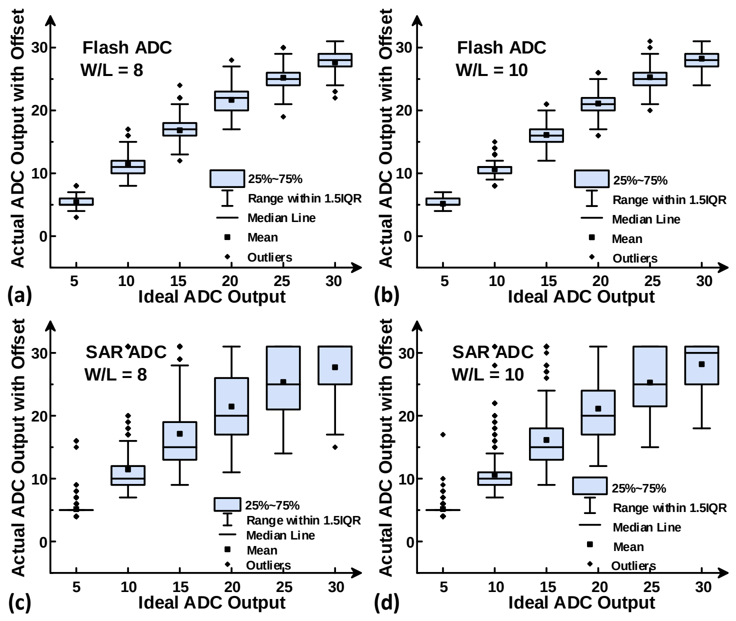


Figure 14 - Simulated ADC output with offset sampled from the Iref distribution.

3.2.2 On-chip fine-tune

According to the previous chapter, ADCs play an important role in CIM's hardware and software performance to accelerate DNN. From the software perspective, full precision ADC is preferred to avoid quantization loss, while low precision ADC is preferred for hardware for low area and energy overhead. As mentioned before, the process variation could introduce ADC offset, causing ADC errors upon ADC quantization loss. As discussed in 1.3.1, the ADC offset could be compensated by adjusting the ADC references or circuit techniques. The former is concerned with generating different references for every ADC on-chip, and the latter will introduce a bigger area overhead. In this work, instead of solving the problem from the ADC side, we utilize the DNN's property of self-adaption to noise.

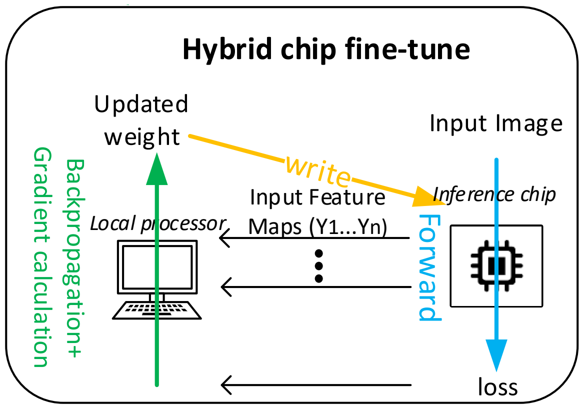


Figure 15 - On-chip fine-tuning dataflow.

Since there are generally many ADCs on-chip, it is time-consuming to read out the exact offset for pure software compensation. Thus, an easier way is the on-chip/off-chip hybrid fine-tuning to include the ADC offset automatically. The basic flow is shown in Figure 15: two copies of weights are obtained in this system, one on-chip and the other in software (local processor). The on-chip inference will be conducted for a specific chip with ADC offset captured, whose output will be used to calculate the loss concerning the real label. The backpropagation will be performed in pure software with a copy of the weights. After that, the weight gradients can be calculated in software with the input feature maps (generated by inference on-chip) and the error feature maps (generated by backpropagation off-chip). The backpropagation and gradient calculation are done in floating-point as they are software-based. Finally, the weights in the software platform and on-chip will be updated by the gradient from a batch of images simultaneously. The weights on-chip can be updated with write-verify to achieve an accurate value.

3.2.3 Performance recovery

The PyTorch platform and NeuroSim framework are used for software and hardware effectiveness evaluation, respectively. We present the VGG-8 network for CIFAR-10 dataset classification with 8-bit activations (input) and 2-8 bit weights. The low precision scheme of inference on-chip is similar to the WAGE algorithm and could get ~92% accuracy for all precision settings. For the hardware-related settings, binary RRAM with Ron=20kΩ and Roff =2MΩ [61] is used for the CIM array. The cell variation is not included since the aggressive write-verify [27] programming scheme is assumed.

Assuming a software-trained network (software baseline) is loaded to chips with ADC variation, accuracy loss will be introduced to the on-chip network, as shown in Figure 16. Comparing the Flash-ADC and SAR-ADC under the same transistor size (W/L), the former will always introduce smaller accuracy degradation. This trend matches the observation that Flash-ADC introduces smaller ADC errors because of level compensation. For the same reason, the inference accuracy will increase as the transistor size increases.

Retrain curves of Flash-ADC and SAR-ADC with different transistor sizes are shown in Figure 17. The Flash-ADC recovers fast for all the W/L under test with small initial accuracy drops. Meanwhile, the fine-tuning effectiveness varies with the precision of the weight and W/L for the SAR-ADC. In all three weight precision settings, when the W/L is too small, the fine-tuning could not fully recover the accuracy due to big variations. The situation will improve as the variation decreases with the increase of W/L. In conclusion, hybrid fine-tuning could compensate for the accuracy loss caused by process variation. Its effectiveness is decided by the ADC typologies and transistor size of the SA.

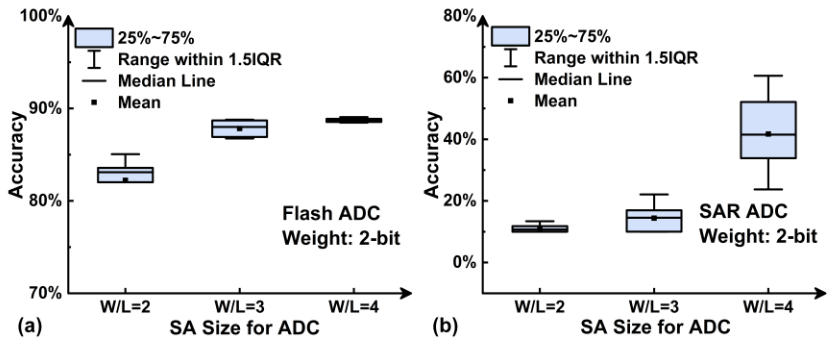


Figure 16 - Accuracy before fine-tuning for (a) Flash-ADC and (b) SAR-ADC.

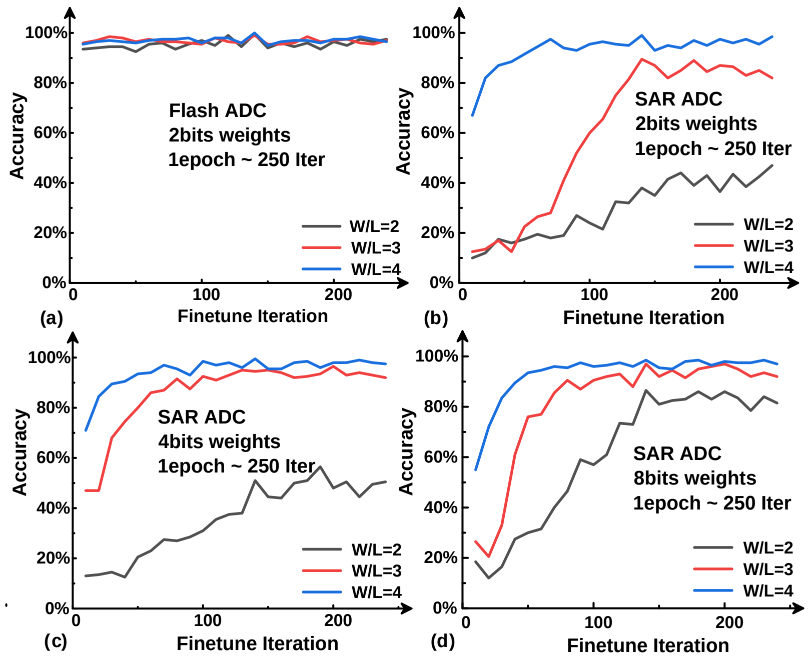


Figure 17 - Retraining curve of (a) Flash-ADC with 2-bit weights; (b) SAR-ADC with 2-bit weights; (c) SAR-ADC with 4-bit weights; (d) SAR-ADC with 8-bit weights.

Unlike training from scratch, fine-tuning here is limited to one epoch considering the overhead introduced. Table 1 list the percentage of changed weight per iteration, namely, the number of cells that need to be programmed. The percentage will increase as the weight bits increase, yet small for all layers and settings, making the overhead of fine-tuning the network acceptable.

Table 1 - Average percentage of weights updated per iteration.

	2bit weight	4bit weight	8bit weight
Conv1	0.127%	0.325%	1.954%
Conv2	0.055%	0.126%	0.835%
Conv3	0.055%	0.105%	0.382%
Conv4	0.053%	0.073%	0.444%
Conv5	0.044%	0.048%	0.280%
Conv6	0.025%	0.024%	0.150%
FC1	0.012%	0.015%	0.077%
FC2	0.038%	0.083%	0.351%
Overall	0.020%	0.025%	0.142%

Table 2 - Hardware overhead for fine-tuning of one epoch.

Weight precision		2bit	4bit	8bit
Feedforward (Inference)	Energy Efficiency(TOPS/W)	13.31	6.78	3.43
	Throughput(FPS)	1644.27	1399.54	1217.61
	Area(mm^2)	53.18	92.20	176.80
	Total Energy Consumption(J)	4.63	9.08	17.94
	Total Latency(s)	30.41	35.73	41.06
Weight update (Retrain)	Total Energy Consumption(J)	0.0015	0.0043	0.0323
	Total Latency(s)	2.12	3.55	6.73

For the hardware evaluation, in the on-chip inference stage, Ron=20k/Roff=2M and 0.2V read voltage are assumed for 128×128 subarrays with 5-bit ADCs. During the weight update stage, write-verify programming with five pulses on average for SET and RESET of the RRAM cell is used with 300ns of 3V write voltage. Table 2 shows the hardware performance reported by NeuroSim with the corresponding settings. Overall, the overhead caused by fine-tuning is acceptable.

3.3 Summary

In this work, we explored ADC offset modeling under process variation and evaluated their effects on the software performance. It has been found that the ADC structure will affect its robustness to the process variation. In detail, the Flash-ADC will see less performance degradation than SAR-ADC under the same variation levels. The process variation will cause bigger ADC offset under smaller SA transistor sizes and thus cause severer software performance loss. Thus, the performance of CIM under process variation could be guaranteed by utilizing big transistors in ADC at the expense of area overhead. As an alternative solution, fine-tuning on-chip is proposed to recover the accuracy loss caused by the ADC offset. The accuracy of the model on-chip could be fully recovered unless the transistor is too small in a bad ADC structure. We also evaluate the overhead of the weight fine-tuning in both time and energy, which is shown to be acceptable in the chip testing phase.

CHAPTER 4. Achieving High In-Situ Training Accuracy and Energy Efficiency with Analog Non-Volatile Synaptic Devices

4.1 Motivation

As illustrated in section 3.2.3, we know that on-chip fine-tuning could help to recover the software performance degradation caused by the ADC offset. Also, training with noise injected can help to eliminate the effect of other non-idealities, such as IR drop [16] and device variations [62]. Thus, if the CIM accelerators could directly support training, these reliability issues could be solved at no expense. On the other side, there is currently no good-for-all model that could work in any circumstance. Thus, the models used in real life tend to be personalized by incremental learning or network fine-tuning, which also prefers the ability to train on-chip for edge devices.

It is relatively easy to equip the SRAM-based CIM with the ability to learn [63] as it is binary and easy to write. On the contrary, using eNVMs for representative DNN models to achieve high in-situ training accuracy remains a grand challenge today [64]. Also, unlike the inference-only chip, training must maintain many intermediate data, making DRAM access unavoidable. The DRAM access could limit the computing efficiency brought by the CIM. This work comprehensively studies these non-ideal effects and seeks possible hardware-aware algorithmic solutions for in-situ training with eNVMs. Additionally, a segmented calculation scheme is used to maintain the performance for training on-chip, showing that the in-situ training with eNVMs is promising, considering both the software and hardware performance.

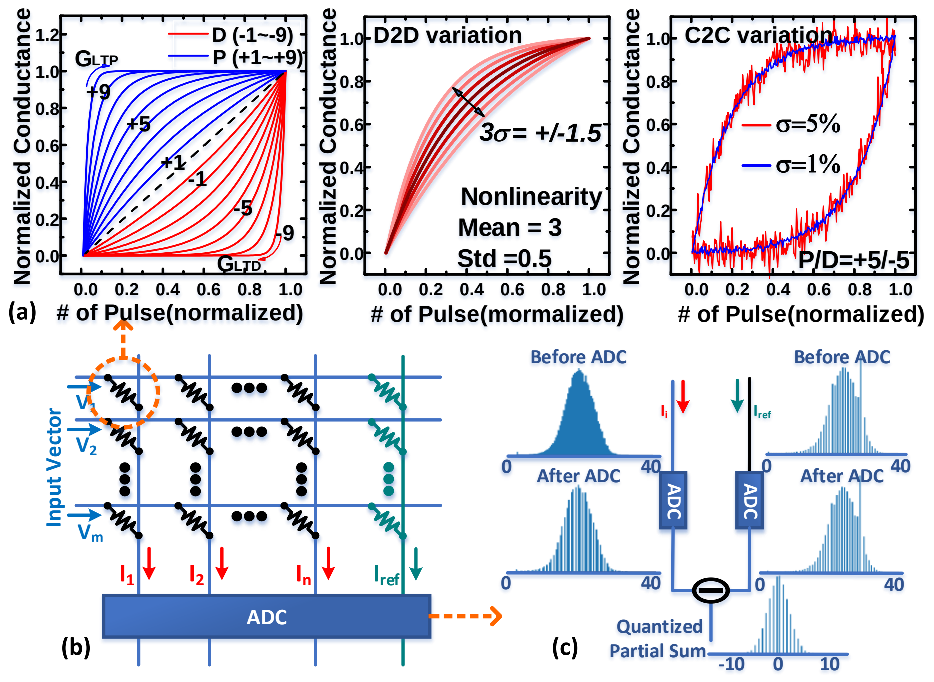


Figure 18 - Non-idealities of analog synaptic devices in (b) eNVM array for in-memory computing with (a) asymmetric and non-linear conductance tuning, device-to-device (D2D) variation, and cycle-to-cycle (C2C) variation and (c) ADC quantization error. A reference column by subtraction is used to represent negative weights.

4.2 Non-idealities for in-situ training accuracy

This work aims to realize the training through stochastic gradient descent (SGD) on the analog eNVM-based CIM platform. The main bottleneck is located in the weight update step. For training on-chip, the desired weight change (ΔW), calculated from the gradient, is mapped to the number of pulses to change the cell conductance (ΔG). Some key challenges of CIM with analog synaptic devices are shown in Figure 18. The eNVM cells introduce non-idealities such as asymmetry/nonlinearity, device-to-device (D2D) variation, cycle-to-cycle (C2C) variation, and a limited number of states. These non-ideal effects could make cell conductance change with a real weight change (ΔW_{ni}) shift from ΔW without the expensive write-and-verify scheme, making the training procedure on the eNVM-based CIM platform different from the software case. Besides, ADCs introduce quantization errors for both forward and backward propagations, making them worse than inference-only engines.

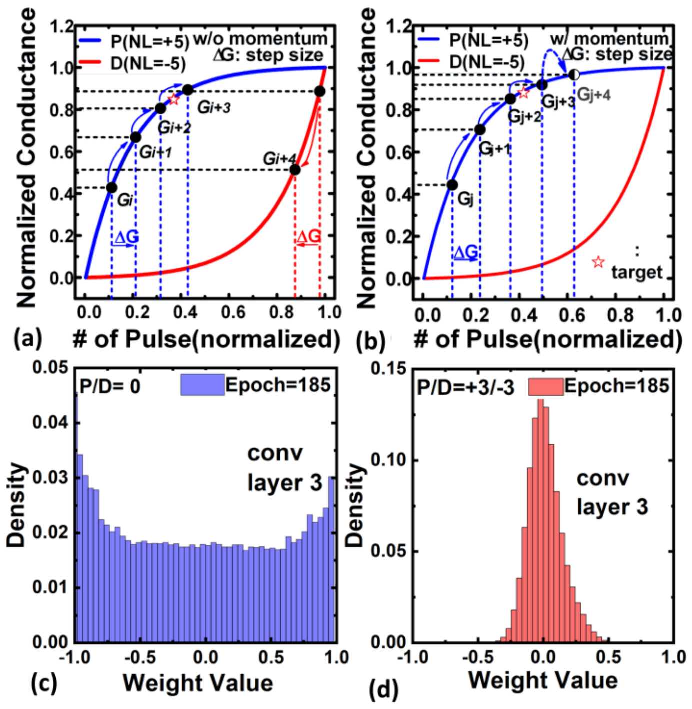


Figure 19 - (a) Conductance update trend without momentum. (b) Conductance update trend with momentum. (c) Weight distribution without asymmetry/nonlinearity. (d) Weight distribution with P/D = +3/-3 for Conv layer 3 for the VGG-8 network at epoch=185.

4.2.1 Asymmetry/nonlinearity in conductance tuning

The asymmetry/nonlinearity implies that the conductance change varies with the current state and the change direction under the same write pulse. Depending on the trajectory's distance from the linear case, a nonlinearity factor (NL) is labeled from 0 to 9. The trends are distinct for potentiation and depression (P/D) and are labeled as +/-, as shown in Figure 18 (a) by the blue/red curves. Demonstrated in the prior work [29], asymmetry is the key fact that causes significant accuracy loss instead of nonlinearity. We propose a hypothesis as an intuitive explanation for this phenomenon. For a memory device approaching the Gmax/Gmin by consecutive positive/negative pulses, a negative/positive pulse (as defined by a sign change in ΔW) will make a large drop/increase in the conductance because of asymmetry/nonlinearity, as shown in Figure 19 (a). As a result, it is statistically easier for the device to return to the middle conductance range than approach Gmax or Gmin. This hypothesis can be validated by comparing the weight distribution trained with P/D=0 and P/D=+3/-3, as shown in Figure 19 (c) (d).

Thus, with asymmetry/nonlinearity, any undesired sign change for ΔW will make the training oscillate and should be avoided. In general, three types of ΔW sign change are undesired in the training process: 1) sampling error of batches; 2) oscillation around local minima; 3) oscillation around global minima (Figure 20).

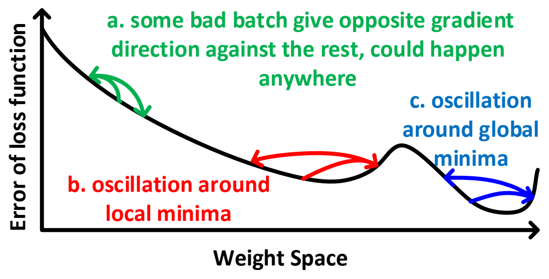

a. some bad batch give opposite gradient direction against the rest, could happen anywhere
c. oscillation around global minima
b. oscillation around local minima
Error of loss function
Weight Space

Figure 20 - Three types of undesired ΔW sign change during training.

To thwart the undesired sign change, we utilize "momentum" by increasing ΔW along the direction of a constant sign, as shown in equation (15). While momentum could eliminate the first two types of undesired sign change, the oscillation around the global minima is unavoidable, whereas it could be mitigated by momentum combined with stochastic quantization. As shown in Figure 19 (a) and (b), without momentum, the direction and probability of weight update are decided directly by the gradient. If the gradient is small, the weight update barely happens. It can be hard to approach Gmax/Gmin since the ΔG is gradually saturated. Moreover, any sign change of gradient could lead to a big jump-back towards the middle conductance. With momentum, it will decide the direction and probability of weight updates. On one side, the momentum will accumulate gradients in the same direction. Thus the probability of going in that direction will be higher and higher, making it easier to approach global minima around Gmax/Gmin. Conversely, when the conductance exceeds the global minima and leads to an opposite gradient, it will first decrease the potentiation/depression probability instead of directly flipping the sign. Although overshooting is also undesired, it will be small because of the saturation nature of the potentiation/depression curve. As a result, momentum with stochastic quantization compensates for the asymmetry/nonlinearity of the conductance tuning.

$$\Delta W(t) = \beta \Delta W(t-1) + (1-\beta) \cdot (-\frac{\partial L}{\partial W}) \tag{15}$$

4.2.2 D2D variation and C2C variation

The NL of P/D could be different from cell to cell for the same type of device, which is defined as D2D variation. Such variation is static, and the DNN model could self-adapt to it. A more severe problem is the C2C variation, which causes ΔG to vary upon each pulse. When the ΔG variation overwhelms the direction that the momentum defines, a positive update step may end up with a decreased conductance, resulting in a loss increase. The C2C variation could be viewed as a temporal noise injected into the weight update, which is tolerable when small. Training will become a random search when the C2C variation is too big. Typical C2C values for some representative devices are shown in Table 3.

Table 3 - Survey of representative analog synapses reported in the literature with weight precision, P/D, and C2C variations.

Analog synapse reported	ΔW precision	C2C	P/D
HZO FeFET [65]	32=5bit	0.5%	1.75/1.46
2T-1FeFET [31]	64=6bit	0.5%	0.85/0.85
2PCM+3T1C [30]	64=6bit	1.5%	0.2/-0.2
Epi-RAM [66]	64=6bit	2%	0.5/-0.5
RRAM [67]	128=7bit	3.7%	0.04/-0.63
ECRAM [68]	1000=10bit	<0.5%	0.347/0.268

4.2.3 Update step size

Although the cell for training is viewed as an analog synapse with continuous conductance, the programming pulse could not be arbitrarily small. In other words, the programming pulse is a lower bound of ΔW for each update. The applicable number of pulses of a certain device generally defines the cell precision for the inference case. However, in the training case, it affects the gradient precision (strictly speaking, the ΔW precision considering the learning rate and the use of momentum). Table 3 surveys representative device technologies ranging from 5-bit to 10-bit. Higher precision is preferred for more precise network fine-tuning.

4.2.4 ADC quantization

Since the synapse for training is analog, even a single cell is high-precision. ADCs' full precision will be big for the summed current of all the rows. In addition, since a signed weight in DNN is mapped to a single cell conductance that is always positive, Case3 number representation from section 2.3.1.2 must be used for this analog synapse. A reference column is adopted to the partial sums back to the zero-centered region. This shift could be done in either the analog or digital domain after ADCs, as shown in Figure 18 (c). This work assumes the latter, as analog subtraction is not easy to accomplish. However, the positively accumulated current may have different means for different DNN layers and have a wider dynamic range than the zero-centered partial sums, making the dynamic range requirement of ADC higher.

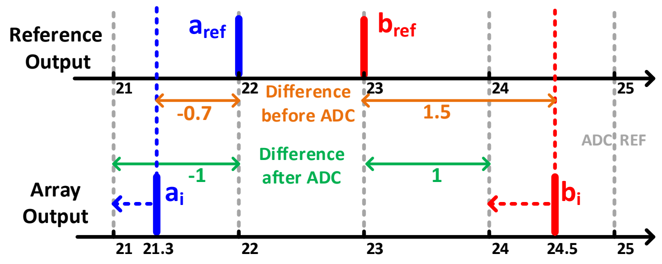


Figure 21 - ADC quantization example.

We use linear ADC quantization for training and set the ADC references for zero quantization bias on the reference column output (I_{ref}). This reference set, combined with the round-down ADC conversion of partial sum, makes the output bias towards negative (Figure 21). Biasing towards negative will deactivate more neurons than it should and prevent training from convergence since many activation functions will cut the negative path, such as ReLU and sigmoid. To solve this problem, round-center quantization could be done by adding +1 to the LSB of all the negative partial sums.

4.3 Training on chip

4.3.1 Training-on-chip architectures

The SGD training process of DNN consists of four steps, namely, feedforward (FF), error calculation (EC), gradient calculation (GC), and weight update (WU). The FF and EC steps could be considered convolutions with the same but transposed weight matrices. Thus, previous works either use two copies of transposed weights [55] or a transposable CIM array to support FF and EC calculation [63]. The latter is preferred for the CIM training platform with analog synapses since it is hard to copy the weights because of the cell variations and limited precision of periphery circuits.

The GC process could also be viewed as a convolution operation between the input feature map (IFM) and error feature map (EFM) from the same image. This convolution could still be implemented in the CIM structure with EFM stored as “weights.” Unlike FF and EC, which always have network weights participate in the convolution for different images, the GC convolution sees different operands for each image. As a result, using eNVM-based CIM in the GC stage is no longer efficient as the “weights” (EFM) need to be frequently updated. Instead, it will be more practical to implement GC with digital circuits or SRAM-based CIM.

Finally, the cell's conductance is fine-tuned based on the gradients from a batch of images in the WU stage. One-time writing is assumed in this work since we take the nonlinearity/asymmetry and variation into account. Although programming overhead for eNVMs is high, the batch-training manner of SGD makes the WU process less dominant in training.

Figure 22 (a) shows a naïve dataflow for training that conducts these four steps one by one. Compared to the inference, DRAM access is unavoidable in training with SGD considering a batch size bigger than one. The IFMs/EFMs for each image must be saved to DRAM in the FF/BP stage and loaded for gradient calculation at the GC stage. Then, the calculated gradients will be saved to DRAM and loaded to the chip in the WU stage for accumulation across different images. Finally, momentum is loaded on the chip, updated by the accumulated gradient, used to update the cell conductance, and then saved back to DRAM for the next iteration. These data movements kill the energy efficiency of CIM in the naïve dataflow.

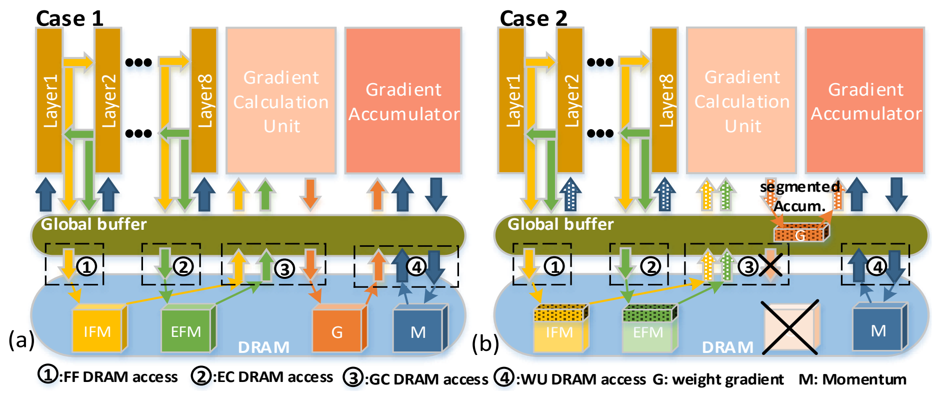


Figure 22 - (a) On-chip training architecture with naïve step-by-step dataflow (case1). (b) On-chip training architecture with segmented on-chip accumulation and updates (case2).

4.3.2 Segmented calculation

To alleviate the massive DRAM accesses in training, a segmented gradient calculation data flow (case2) is proposed, as shown in Figure 22 (b). The new scheme adopts the same flow for FF and EC, making no difference in DRAM access for IFMs ① and EFMs ②. However, the GC and WU are interleaved (Figure 22 (b) step ③) to calculate the gradient and update the network's weights segmentally. In more detail, as there is no need to update the weights of the whole network at once, gradients from one layer will be calculated, accumulated, and used to update the momentum to tune the cell conductance part by part. The part will be small enough so that its gradients can be held on-chip in the global buffer, eliminating the DRAM access of the gradient.

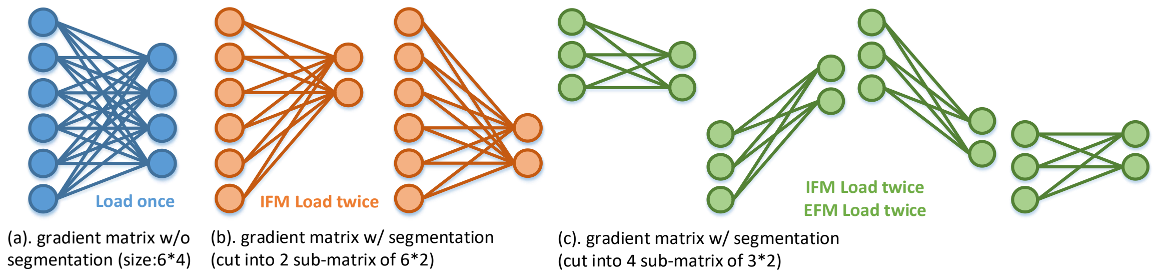

Load once
IFM Load twice
IFM Load twice
EFM Load twice
(a). gradient matrix w/o segmentation (size:6*4)
(b). gradient matrix w/ segmentation (cut into 2 sub-matrix of 6*2)
(c). gradient matrix w/ segmentation (cut into 4 sub-matrix of 3*2)

Figure 23 - Example of input and error feature map reload for case2 segmented gradient calculation.

One concern of the segmented gradient calculation is the reloading of IFMs/EFMs. In the naïve dataflow, the IFMs/EFMs need only be loaded once back to the chip for the gradient calculation. However, if the layer's gradient matrix is cut into pieces, some feature maps (IFMs and/or EFMs) may need to be reloaded, as shown by an example in Figure 23. Then, there will be a trade-off between DRAM access for FMs and DRAM access for gradients. Generally, if the weight size is much larger than the IFM/EFM, such as in FC and deep CONV layers, reloading FMs is more economical. Otherwise, it will be better to allow gradient access, taking shallow CONV layers as an example.

To cover all cases, we consider a hybrid scheme that optimizes the GC and WU flow layer wisely, which means the one with less DRAM access between naïve flow and segmented calculations will be applied to each layer. An example of the number of DRAM access across different layers in the VGG-8 network for CIFAR-10 classification is shown in Figure 24. As expected, case1 overwhelms case2 in the shallow layer when the global buffer is small, while case2 always wins for deep layers. As global buffer size increases, case2 will be better as the gradients of the whole layer could be fitted into the global buffer.

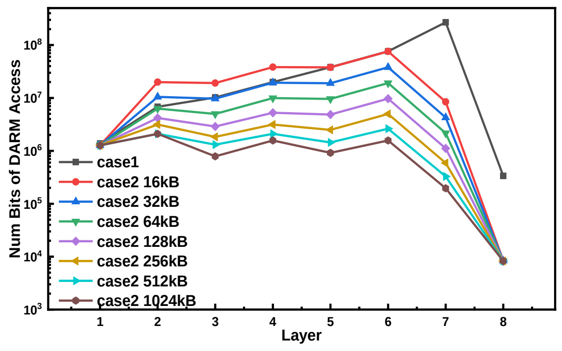


Figure 24 - The number of DRAM access bits for each layer under different global buffer sizes from 16 kB to 1024 kB for case1 (naïve) and case2 (segmented gradient calculation) schemes.

4.4 Evaluation and discussion

We evaluate the analog-synapse-based CIM platform using the VGG-8 network training on CIFAR-10 classification. From the training perspective, we assume inputs (IFM), errors (EFM), and gradients of the CONV/FC layer are quantized while weights are floating-point (analog cell). In the following content, we will no longer emphasize that weights are floating-point in training. Instead, we use weight precision to imply gradient precision to keep consistent with inference. We modify WAGE [46] to integrate all the non-idealities of eNVM-based CIM. The default setting uses 8-bit weight (gradient) and 8-bit IFM/EFM from the hardware point-of-view. Without specification, nonlinearity/asymmetry is analyzed with P/D=+3/-3. The ADC quantization is introduced at the edge of each 128x128 array except for the first layer. With a batch size = 200, the ideal software baseline could achieve 92% using WAGE quantized training.

4.4.1 Software evaluation

We first evaluate the effectiveness of momentum on nonlinearity/asymmetry compensation. Training results with different momentum factors for P/D=+3/-3 and P/D=0/0 are shown in Figure 25 (a). With β equals 0.9, the accuracy of P/D=+3/-3 is significantly boosted while P/D=0/0 is slightly affected, validating that momentum could degrade the accuracy loss caused by nonlinearity/asymmetry. Consequently, 0.9 is used for β in the rest simulations. Afterward, in Figure 25 (b), various NL factors are tested under asymmetry. By fine-tuning the learning rate to control the number of weights updated for each iteration, ~87% accuracy is still achievable up to extreme P/D=+9/-9.

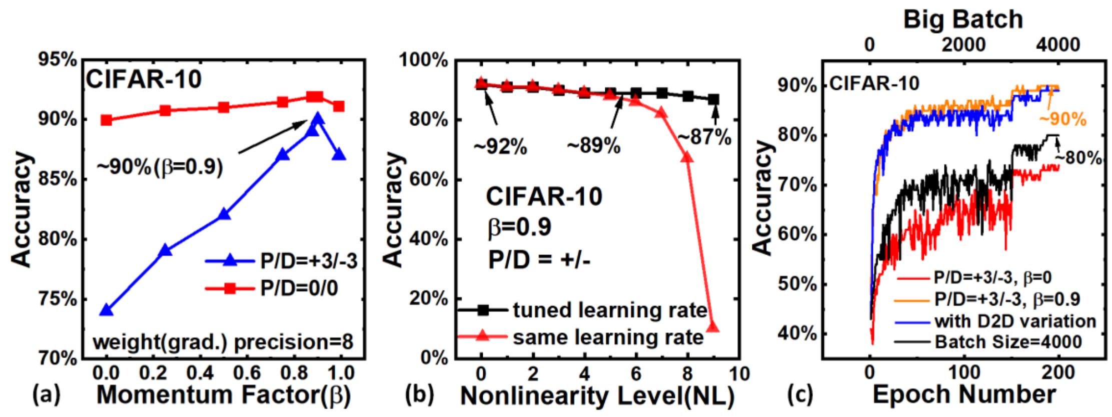


Figure 25 - (a) Accuracy vs. momentum factor β. (b) Accuracy vs. device nonlinearity factor under asymmetry. (c) Training traces w/wo momentum, D2D variation and big batch size=4000.

To further validate our hypothesis of undesired sign change, we evaluate the training process with a very big batch size. In this case, the batch sampling noise will be diminished. As shown in Figure 25 (c), a batch size = 4000 is used to train the network with P/D=+3/-3. Since the batch size increases, the number of epochs is enlarged correspondingly to achieve the same number of weight updates. We could see from the result that the big batch size could achieve higher accuracy than the smaller one under P/D=+3/3 and $\beta = 0$. But still, it can not get a result as good as the momentum solution. The same figure shows the training result with a D2D variation around P/D=+3/-3 with a 0.5 standard deviation by the blue curve. The trend is very similar to the P/D=+3/-3 case without variation, showing that D2D variation is not a concern if nonlinearity/asymmetry is no longer a problem.

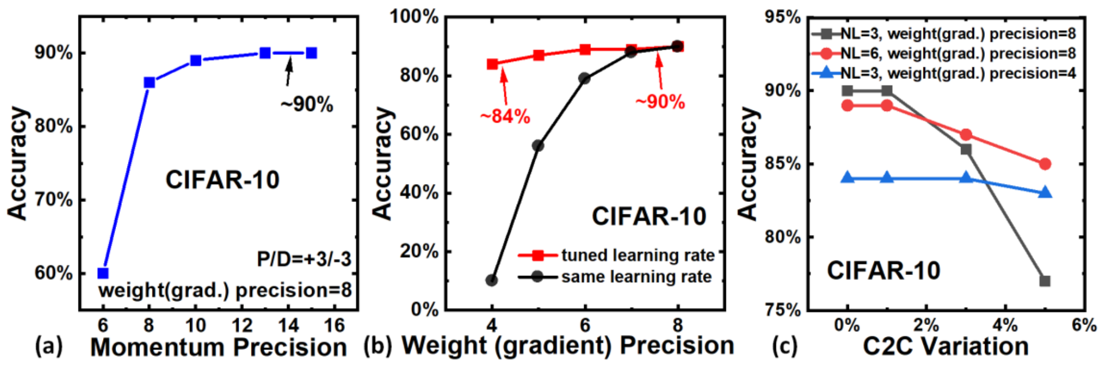


Figure 26 - (a) Accuracy vs. momentum precision. (b) Accuracy vs. device weight (gradient) precision. (c) Accuracy vs. C2C variation. Momentum β =0.9 is applied.

We also apply quantization on momentum to reduce the memory overhead to hold it and the energy overhead caused by loading/storing it for each WU iteration. Unlike gradients, momentums need to be quantized statically instead of stochastically, introducing a higher precision requirement. Figure 25 (a) shows that 10~12 bits are required for momentums to avoid accuracy loss.

Then, we evaluate the cell precision effect on training. As mentioned before, cell precision defines the updating step size in training—the lower the precision, the bigger the step. Surprisingly, even down to 4-bit weight (gradient) is still feasible under fine-tuned learning rate, as shown in Figure 26 (b).

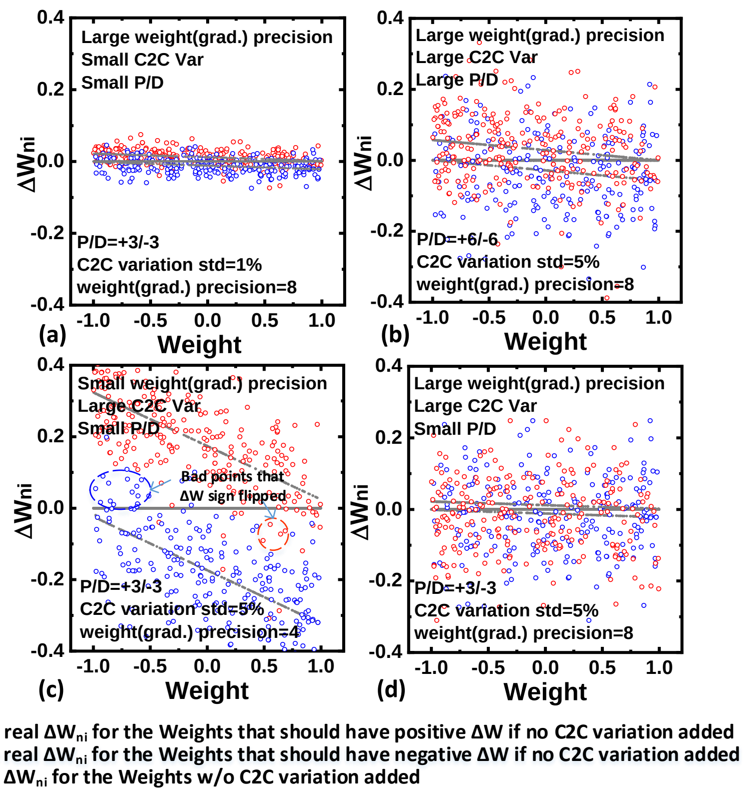


Figure 27 - Statistical weight update pattern (ΔW vs. W) in one batch for different P/D, C2C variance, and weight (gradient) precision. (a) When C2C variation is small for devices with small P/D and high precision. (b) When C2C variation is large for devices with large P/D and high precision and case (c) When C2C variation is large for devices with small P/D and low precision.

We find that the effect of C2C variation highly depends on NL level and weight precision. Since the drawback of C2C variation is to flip the desired shift direction of the conductance, a big update step will have a lower probability of being exceeded by the variation. We can see this from the results shown in Figure 26 (c). Low precision and high NL devices are more robust as C2C variation increases since they imply greater conductance change. To better understand this, we check a batch of ΔWni vs. W with different NL, C2C, and cell precision. As shown in Figure 27, the color denotes the weight update direction according to the gradient. The red ones need an increase while the blue ones should be decreased. The y-axis of each dot means the real weight change mapped from conductance change with nonlinearity/asymmetry and C2C variation. The desired weight update direction is flipped when a red/blue dot has a negative/positive ΔWni. We could see that bigger NL, smaller weight (gradient) precision, and smaller C2C variation are desired for fewer cross-over points.

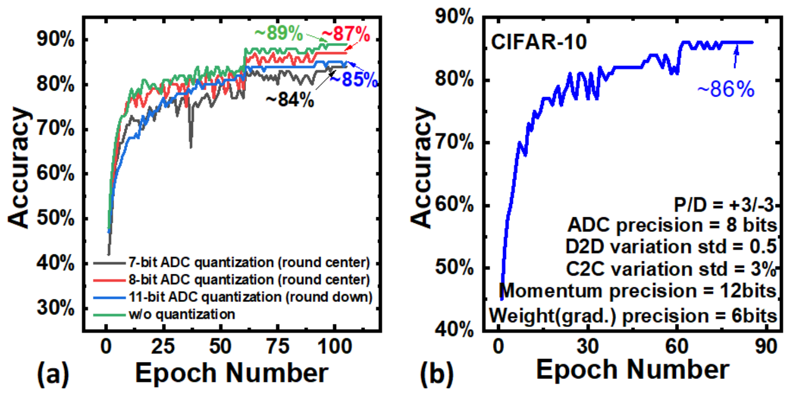


Figure 28 - (a) Training traces for different ADC resolutions. (b) Training traces when asymmetry/nonlinearity, D2D variation, C2C variation, momentum, and ADC quantization effects are combined.

Ultimately, we explore the ADC requirements for training with analog synapses. If round-down quantization is used instead, the accuracy is only 85%, even with 11bit ADC. By using the round-center quantization, ADC precision could be reduced to 8bit with ~87% accuracy.

In summary, we train a network with all the non-ideal effects combined with the momentum and ADC quantization. A remarkable result of 86% accuracy for in-situ training with practical eNVMs is still achievable, as shown in Figure 28 (b).

4.4.2 Hardware evaluation

While in-situ training with analog synapse is promising from the software side, it is not applicable if the hardware overhead is high. Thus, we evaluate the hardware performance with the NeuroSim+DNN v2.1 platform.

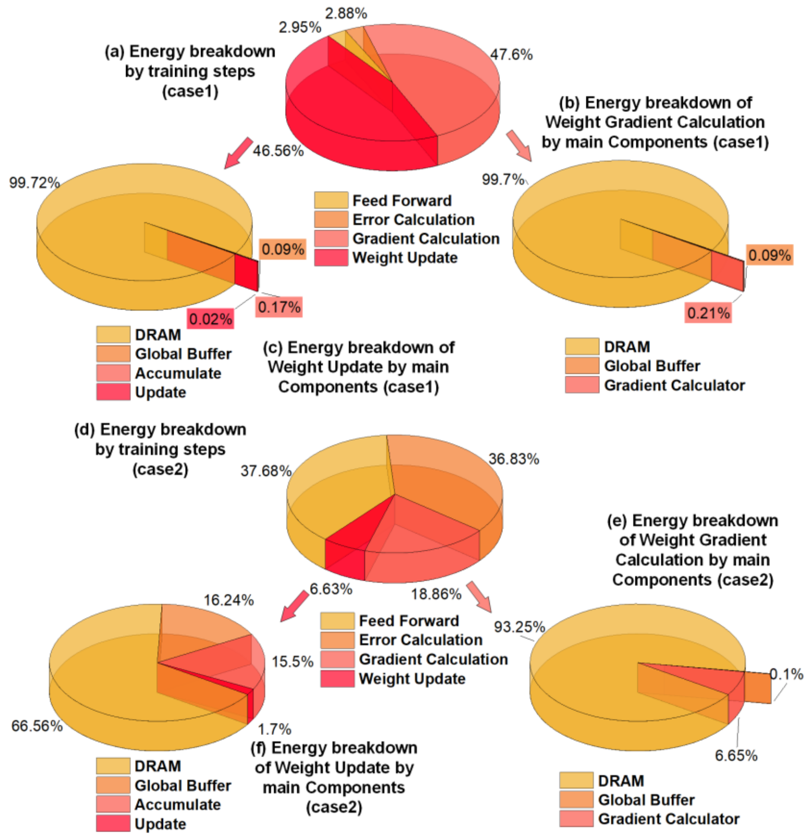


Figure 29 - (a-c) Energy breakdown for case1 (naïve) dataflow. (b-f) Energy breakdown for case2 (segmented calculation) dataflow.

Figure 29 compares the energy consumption breakdown of the naïve scheme (case1) and the segmented gradient calculation scheme (case2) with a global buffer big enough to hold the gradient of the largest layer. The FF and BP energy consumptions are the same for these two cases. From Figure 29 (a) and (d), we can see that the GC and WU processes consume most of the energy in case1 while the corresponding parts are much smaller in case2. Figure 29 (b) and (c), which report the energy breakdown for GC/WU separately, show that the energy consumed by DRAM access takes ~99% due to the gradients and FMs load/saving. For the similar breakdown in Figure 29 (e) and (f) for case2, although the DRAM access energy is still dominant due to the IFMs/EFMs, it reduces significantly as the gradient is eliminated.

A corresponding area breakdown comparison is shown in Figure 30. The difference only lies in the global buffer size. We could see that the DRAM energy reduction of the demonstrated case2 is at the terrible expense of the global buffer size and thus greatly increases the total area.

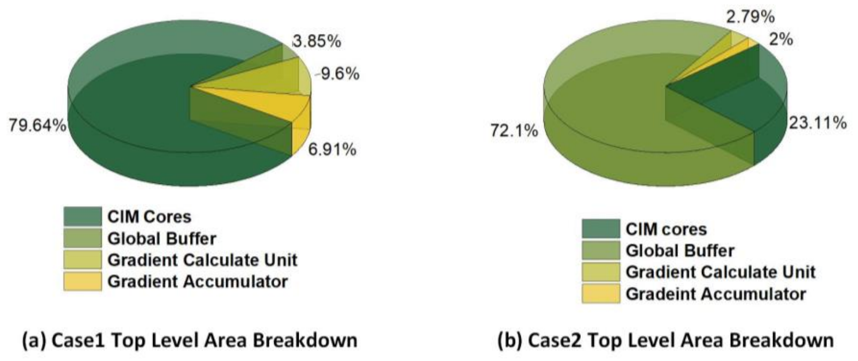


Figure 30 - Area breakdown for (a) case1 (naïve) dataflow (b) case2 (segmented calculation) dataflow.

To find a balanced trade-off point of energy consumption reduction and the area overhead, we plot the energy efficiency and chip area concerning the global buffer size for case1, case2, and the hybrid scheme as mentioned in 4.3.2 in Figure 31. The case2 and the hybrid scheme always beat case1 regarding energy efficiency. Due to the shallow layers, the hybrid scheme is slightly better than case2 for a very small global buffer. The difference fades away with the increase of the global buffer size. As the global buffer size increase, the increase of energy efficiency gradually saturates while the growth of chip area is exponential. Thus, the global buffer size is selected corresponding to the highest energy efficiency/area ratio, which is around 576 kB for the VGG-8 network example.

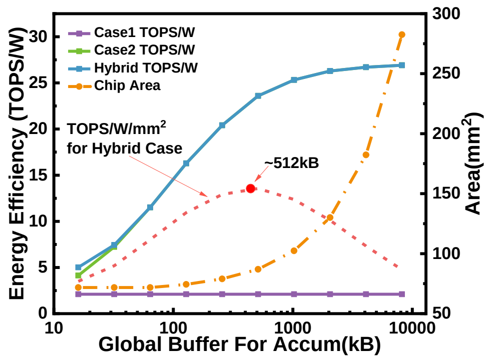


Figure 31 - Energy efficiency vs. global buffer size for case1, case2, and hybrid scheme.

Since the efficiency of the hybrid scheme of case1and case2 is highly dependent on the network structure, a benchmark of hardware performance of different DNN models is presented in Table 4. Four divergent configurations are picked to compare: 1) small feature maps + small networks (CIFAR-10 + VGG-8), 2) big feature maps + big networks (ImageNet + VGG-16), 3) small feature maps + big networks (CIFAR-10 + ResNet-18), 4) big feature maps + small networks (ImageNet + ResNet-18). The hybrid scheme achieves higher energy efficiency than the naïve design for all four cases. As the segmented gradient calculation scheme prefers a bigger gradient size than the IFM/EFM size, configurations 1 and 3 see a greater improvement than configurations 2 and 4. From the area side, the global buffer in case 1 is set by the biggest FM, while the highest efficiency/area ratio decides the hybrid case. The area increase for configurations with big IFM/EFM (2, 4 with IMAGENET) is smaller than the small ones (1, 4 with CIFAR-10) since their starting point is already high.

Table 4 - Hardware performance benchmark (Training).

Technode: 40nm			ADC:8bit		
Ron/Roff: 100K/1M			Readvoltage:0.5V		
Write Pulse Voltage: 1.6V/1.5V			Write Pulse Width: 50ns/50ns		
Network		VGG-8	VGG-16	ResNet-18	
Parameter Size		~13M	~138M	~11M	
dataset		CIFAR-10	ImageNet	CIFAR-10	ImageNet
ArraySize		128x128	64x64	64x64	64x64
Input Size		32x32	224x224	32x32	224x224
Case1	TOPS/W	2.11	3.63	0.15	4.39
	Global Buffer	128kB	3136kB	64kB	784kB
	Area(mm^2)	71.52	1014.19	99.35	292.38
Hybrid	TOPS/W	23.59	11.10	7.74	10.408
	Accu. Buffer	512 kB	2048 kB	256 kB	512 kB
	Global Buffer (Accu. Buffer)	576kB (512kB)	3616kB (2048kB)	288kB (256kB)	904kB (512kB)
	Area(mm^2)	86.995	1033.716	110.233	296.997

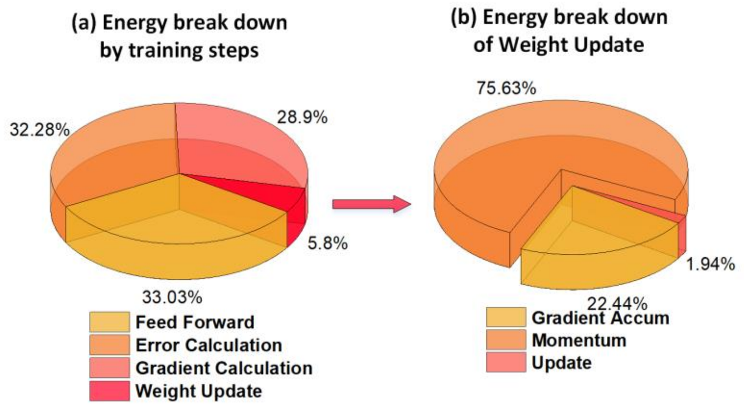


Figure 32 - Energy breakdown for hybrid dataflow with the portion of momentum in the weight update highlighted.

As the software performance section discussed, momentum is important for high in-situ training accuracy. While the segmented gradient calculation scheme could eliminate the off-chip talk of the gradient, the DRAM access of momentum is unavoidable. In the WU stage, the momentum is loaded to the chip, recalculated with the gradient, used for weight update, and then saved back for the next iteration. Figure 32 shows the energy breakdown of training with the hybrid scheme assuming a 572 kB global buffer (512kB for accumulation) to highlight the momentum overhead. If we only consider the WU step, the energy consumption by loading/saving momentum is dominant, as shown in Figure 32 (b). However, from Figure 32 (a), the WU part is small among all four steps, thanks to the batch operation. As the batch size decreases, the number of WU will increase for one epoch, introducing more momentum transfer. As a result, the WU will be more dominant and significantly decrease energy efficiency (Figure 33). Moreover, a small batch size will introduce more batch sampling errors, which is unfriendly to the analog synapses with nonlinearity/asymmetry. Thus, the proper batch size is desired from both the software and hardware point-of-views.

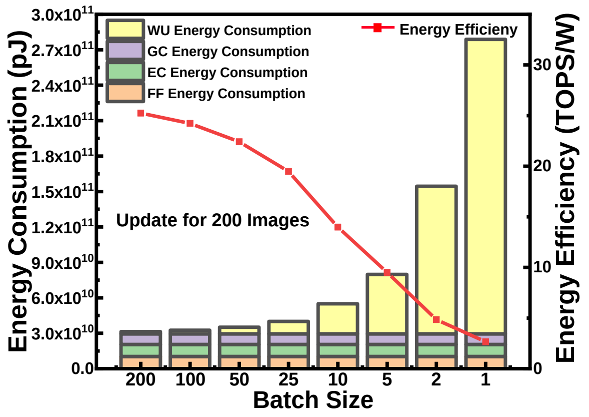


Figure 33 - Energy consumption & energy efficiency vs. training batch size.

4.5 Summary

We explore the reliability of in-situ training with eNVMs for embedded AI platforms in this work. First, momentum solves the critical bottleneck for training brought by asymmetry/nonlinearity and D2D variation. High cell precision is not urgent but is still preferred. C2C variation is tolerable to some degree based on the devices' properties.

Further device engineering to suppress it is still desirable, especially for high-precision cells. ADC resolution needs to be carefully considered for good training performance. For the VGG-8 training for CIFAR-10 classification, 86% accuracy is achievable under non-idealities, including P/D=+/-3 asymmetry and nonlinearity, 0.5 D2D variation, 3% C2C variation, and 8bit ADC quantization. This result suggests that the training with eNVMs is still reliable under device and circuit non-idealities.

This work also discusses the training architecture for the CIM with analog synapses. With optimized dataflow by segmentally performing gradient calculation and weight update, the high energy brought by CIM could still be maintained for training. Energy efficiency 25.24 TOPS/W was reported for training VGG-8 on the CIFAR-10 dataset. With a properly chosen batch size for training, the hardware cost of the introduced momentum training could be minimized to < 4.2% energy overhead on-chip.

CHAPTER 5. Secure XOR-CIM Engine: Compute-in-Memory SRAM Architecture with Embedded XOR Encryption

5.1 Motivation

In a typical scenario, the training of the DNNs is done in the cloud, which does not worry about energy consumption and resource availability. After that, edge devices download the well-trained model for inference. The CIM has been proven to be a good choice for such edge devices. However, new security challenges are raised by this communication, such as the DNN model leaking. As is known, a well-trained model could be a business property or preserve sensitive information, making its protection urgent.

Encryption is an elementary choice to protect sensitive data in communication. However, decryption could be difficult on lightweight edge devices, especially for big neural networks. Also, for CIM-based edge devices, the balance between storing the data in an encrypted format and conducting parallel operations on raw data is challenging.

This work has developed a lightweight yet efficient countermeasure to protect DNN models in a two-party system of the cloud and the SRAM-based CIM accelerated inference engines. The overhead brought by the protection is shown to be small for all aspects concerning throughput, energy efficiency, and area.

5.2 Secure XOR-CIM inference engine

5.2.1 Secure inference engine system

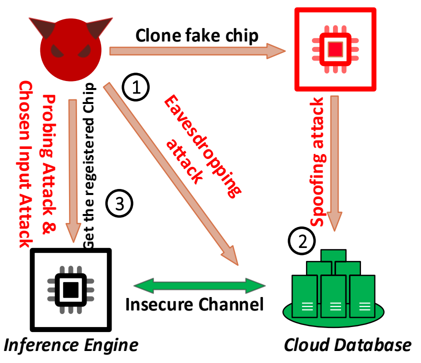


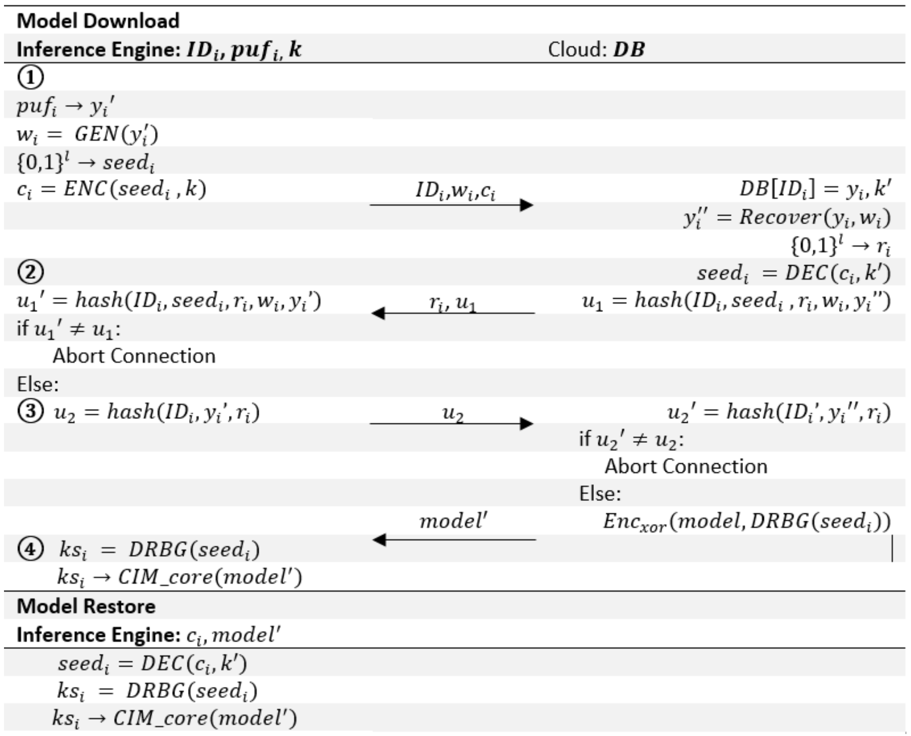
Figure 34 - The two-party system of the CIM inference engine and the cloud database, and the possible threats of model leaking on the system.

The two-party system studied in this work consists of an SRAM-based XOR-CIM inference engine and a cloud database with well-trained models, as shown in Figure 34. The inference engine needs to download the models from the cloud, but the two parties' communication channel is assumed to be insecure. Thus, this model is under the threat of the eavesdropping attack (Figure 34 ①), which gathers sensitive information by monitoring the unprotected channel. Encryption could be used as a defense against this kind of attack. However, edge devices' limited power & resource and the increasing DNN model size make the decryption overhead a considerable problem for the DNN inference engine. Thus, to protect the DNN model in transmission, XOR encryption is hopeful as the hardware overhead caused by its decryption is tiny. While being hardware friendly, the protection brought by XOR encryption could be very weak in a bad protocol. Theoretically, one-time-pad could not be cracked as the messages are XORed with truly random keys. Sharing a key between two messages will immediately degrade the security brought by XOR encryption. However, generating truly random keys and distributing them between the two parts is difficult. Instead, the stream cipher is better by applying XOR encryption with a pseudo-random sequence as the key. In this case, only a random seed needs to be generated and shared between two parties, which is much shorter than the key. A key generating and distribution protocol is used for our two-party system like this: A random seed is first generated on-chip at the inference engine utilizing the SRAM power-on states as an entropy source. This seed is encrypted and sent to the cloud. The cloud uses this seed to generate a pseudo-random sequence to encrypt the DNN model. Then, the encrypted model is transmitted to the inference engine. Simultaneously, the same pseudo-random sequence is generated on-chip on the inference engine and will be used for further CIM calculation.

The countermeasure used for the eavesdropping attack leaves the adversaries a space to apply the spoofing attack by pretending to be a good user to require information from the cloud (Figure 34 ②). Thus, authentication is necessary for building communication. Strong PUF with tremendous challenge-response pairs is a popular choice for authentication from the hardware perspective. While the SRAM on-chip could also be used as a PUF, it is a weak PUF with very limited challenge-response pairs. Luckily, a weak PUF-based authentication protocol is proposed in [69], which we modified to support the inference engine authentication in our system.

Finally, a specific problem with portable devices is that anyone can physically access them. Thus, the adversary could get a registered device and fool the system if only the hardware information is used in the authentication (Figure 34 ③). Considering the device could be stolen, some user-aware-only information is included in our modified protocol. In this work, the key used to encrypt the seed for stream cipher is assumed user-aware-only. The cloud has a corresponding key exchanged in a secure channel in advance for correct decryption. In this case, for the adversary who does not know the key, the decrypted seed on the cloud will not be wrong, and the seed on-chip could not decrypt the model correctly.

Instead of a power-down device, the adversary may get a chip with a model already downloaded in an idle state. We do not assume the model on-chip is wiped during the idle state to avoid big download overhead. However, considering the security issue, the key sequence could not be held on-chip. Only the encrypted seed is saved on-chip. The seed is decrypted when the chip returns to work, and the pseudo-random key sequence is regenerated. The adversary could not get the seed as he/she does not know the key. He/she could neither do the probing attack and chosen input attack on the model as the model saved on-chip is encrypted. In this work, we do not assume the adversary could get a working inference obsessed by the authorized user as it is too strong.



Model Download		
Inference Engine: $\boldsymbol{ID_i, puf_i, k}$		Cloud: $\boldsymbol{DB}$
①		
$puf_i \rightarrow y_i'$		
$w_i = GEN(y_i')$		
$\{0,1\}^l \rightarrow seed_i$		
$c_i = ENC(seed_i, k)$	$\xrightarrow{ID_i, w_i, c_i}$	$DB[ID_i] = y_i, k'$
		$y_i'' = Recover(y_i, w_i)$
		$\{0,1\}^l \rightarrow r_i$
②		$seed_i = DEC(c_i, k')$
$u_1' = hash(ID_i, seed_i, r_i, w_i, y_i')$	$\xleftarrow{r_i, u_1}$	$u_1 = hash(ID_i, seed_i, r_i, w_i, y_i'')$
if $u_1' \neq u_1$:		
Abort Connection		
Else:		
③ $u_2 = hash(ID_i, y_i', r_i)$	$\xrightarrow{u_2}$	$u_2' = hash(ID_i', y_i'', r_i)$
		if $u_2' \neq u_2$:
		Abort Connection
		Else:
	$\xleftarrow{model'}$	$Enc_{xor}(model, DRBG(seed_i))$
④ $ks_i = DRBG(seed_i)$		
$ks_i \rightarrow CIM\_core(model')$		
Model Restore		
Inference Engine: $c_i, model'$		
$seed_i = DEC(c_i, k')$		
$ks_i = DRBG(seed_i)$		
$ks_i \rightarrow CIM\_core(model')$		

Figure 35 - Communication protocol for the secure SRAM-based CIM inference engine.

The protocol we used is summarized as shown in Figure 35. Compared to the original reverse secure sketch [69] protocol, the random number used to defend the replay attack is replaced by our random seed and thus is encrypted instead of transmitted in raw format. The cloud database saves the weak PUF response y_i and key to decrypt the seed k' in advance. In the end, the additional steps for model transmission and restoration are added.

5.2.2 *Secure XOR-CIM inference engine architecture*

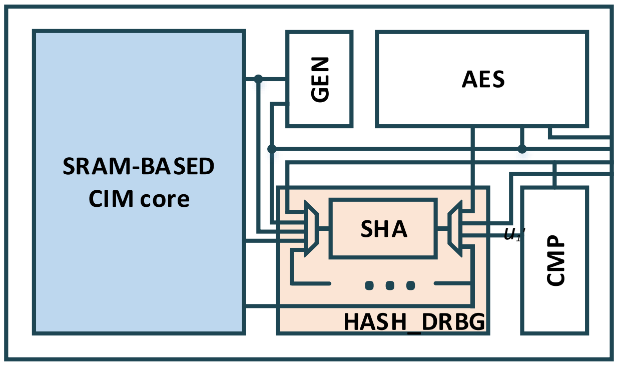


Figure 36 - Secure XOR-CIM inference engine architecture.

The inference engine architecture is designed as shown in Figure 36. It mainly consists of five parts: 1) A SRAM-based CIM core for inference; 2) A generator (GEN) for side information generation for the PUF response; 3) a HASH-based deterministic random bit generator (DRBG) module for pseudo-random number generation; 4) an Advanced Encryption Standard (AES) module for seed encryption/decryption and 5) a comparator (CMP) to check if two messages are the same. It needs to be highlighted that the SHA256 block in the DRBG module is reused for three purposes in the protocol. Firstly, it will be used for the true random seed generation that compresses the power-on states of SRAM cells as the entropy source. Then, it is used as a simple HASH function, as the protocol requires. Finally, it serves as part of the DRBG for pseudo-random bit sequence generation. This reuse could reduce the area overhead brought by the secure protocol.

5.3 Hardware implementation of XOR-CIM core

The heart of our inference engine is the CIM core. As described by the protocol, the model downloaded and saved in idle mode is encrypted with the XOR-based stream cipher. To perform correct VMM operations, the plaintext of the weights should be used for calculation. One method to deal with it is to decrypt the model before using it for inference, resulting in raw data saved in the memory. In this case, the model must be read out and encrypted again if it enters the idle mode, introducing additional overhead. Another method is always to save the encrypted model in memory but to adopt near-memory calculations with decryption, losing the parallelism brought by CIM.

This work proposes an architecture to support CIM directly on the model encrypted by the XOR-based stream cipher, named XOR-CIM. XOR-CIM utilizes the modified dual word lines (WLs) 6T SRAM to apply the key on the inputs instead of decrypting the weights directly.

5.3.1 6T Dual-WL SRAM bit-cell

As mentioned before, we apply the key bits on the inputs instead of decrypting the weights. Assuming weight bits encrypted by key bits, the straightforward way to conduct VMM is to decrypt them first, as shown in equation 15. By playing a small trick (equation 16), the product-and-sum of inputs and decrypted weights could be converted to the product-and-sum of encoded inputs and encrypted weights.

$$\boldsymbol{w_e} = \boldsymbol{w} \oplus \boldsymbol{k}$$

$$Partial\ Sum = \sum In \cdot (w_e \oplus k) \tag{16}$$

$$\boldsymbol{Partial\ Sum} = \sum \boldsymbol{In} \cdot \left(\overline{\boldsymbol{w_e}} \cdot \boldsymbol{k} + \boldsymbol{w_e} \cdot \overline{\boldsymbol{k}}\right)$$

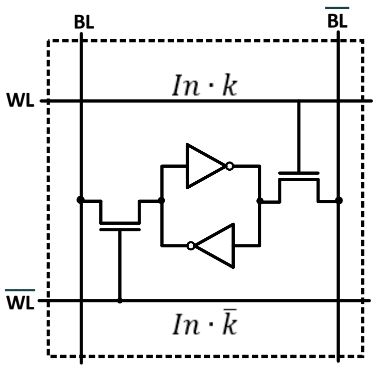
$$= \sum (\boldsymbol{In} \cdot \overline{\boldsymbol{w_e}} \cdot \boldsymbol{k} + \boldsymbol{In} \cdot \boldsymbol{w_e} \cdot \overline{\boldsymbol{k}})$$

$$= \sum (\boldsymbol{In} \cdot \boldsymbol{k}) \cdot \overline{\boldsymbol{w_e}} + \sum (\boldsymbol{In} \cdot \overline{\boldsymbol{k}}) \cdot \boldsymbol{w_e} \tag{17}$$

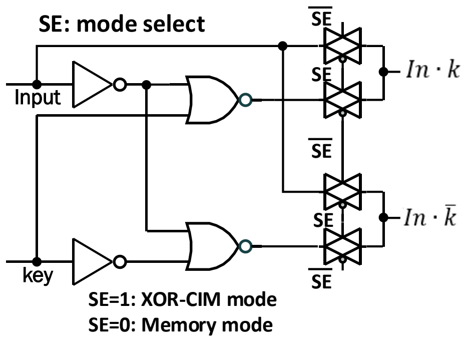
In the reformed calculation, the $\boldsymbol{w_e}$ and $\overline{\boldsymbol{w_e}}$ are directly used in the product-and-sum operation while the input $\boldsymbol{In}$ is modulated by the key $\boldsymbol{k}$. Owing to the latch structure of SRAM, the $\boldsymbol{w_e}$ and $\overline{\boldsymbol{w_e}}$ are naturally available when the encrypted weights are saved. The original product-and-sum is divided into two parts, utilizing either side of the SRAM column separately for CIM operation. However, these two parts see different inputs since one side takes $\boldsymbol{In} \cdot \boldsymbol{k}$ while the other side takes $\boldsymbol{In} \cdot \overline{\boldsymbol{k}}$. That is why the dual WL cells, as shown in Figure 37 (a), are required in this work. One drawback of this structure is that the weights in the same row must share the key bit as their input is shared for parallelism. A discussion will be presented later to prove that this key reuse will not degrade the protection strength of XOR-CIM in practice.

5.3.2 Decrypting input generator

There are two modes that the XOR-CIM needs to work in: the memory mode and the XOR-CIM mode. The encrypted data is written into the array in memory mode. Thus the Dual-WL SRAM cell should work in the same way as the conventional 6T SRAM, which means the WL and $\overline{\mathrm{WL}}$ should work in the same phase: both be “1” or “0”. In the CIM, the inputs to WL and $\overline{\mathrm{WL}}$ are encoded by the k and $\bar{k}$ separately. When the In is “0”, both inputs are "0". But when the In is "1", they could not both be "1" because of the complimentary k and $\bar{k}$. The circuit presented in Figure 37 (b) is used to generate the input signal to support both modes.



(a)



(b)

Figure 37 - (a) Dual-WL 6T SRAM cell. (b) Decrypting input signal generator.

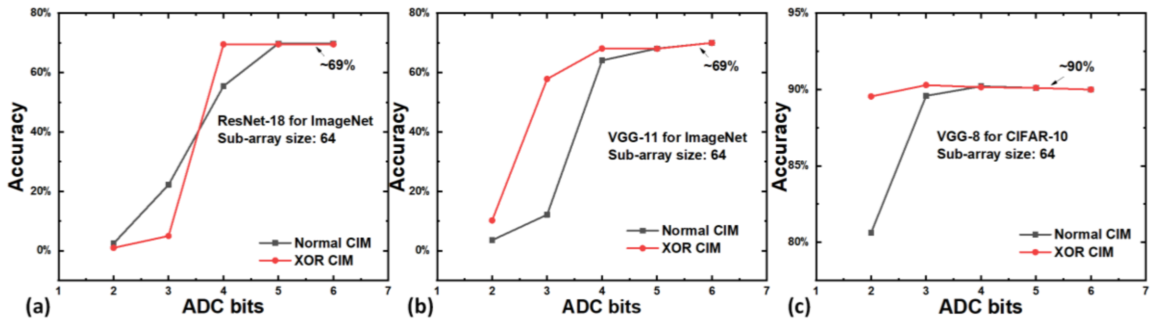


Figure 38 - Inference accuracy vs. ADC resolution.

5.3.3 ADC pair resolution

Compared to the Normal-CIM, the XOR-CIM uses both sides of SRAM for CIM operation. Consequently, a pair of ADCs are needed for each column, which seems to double the number of ADCs for one array. However, by checking the accuracy concerning ADC precision, we find that to achieve negligible software performance loss caused by ADC quantization loss, Normal-CIM needs one more bit for the single ADC than the ADC pair used in XOR-CIM. Since the area and energy consumption brought by ADC is exponentially proportional to the precision, the ADC overhead of these XOR-CIM and Normal-CIM should be similar. This trend is proven on ImageNet classification with ResNet-18 and VGG-11, and CIFAR-10 classification with VGG-8, as shown in Figure 38.

5.3.4 Dynamic reference array

The Dual-WL 6T SRAM cell works the same way as the 6T SRAM for CIM operation. Namely, inputs are applied on the access transistor, and the data at the storage nodes represent weights. Thus, it suffers from the same read disturbance and input-dependent analog output shift problem mentioned in 1.3.1. Low WL access voltage is used to avoid fast voltage drop on BL/$\overline{\text{BL}}$ while the one-side access nature of Dual-WL 6T cell could further reduce read disturbance. Inspired by [25], a dynamic reference array is adopted to solve the input-dependent analog output shift problem.

5.4 Evaluation and discussion

5.4.1 Methodology and setup

We tested ResNet-18, VGG-11, and VGG-8 models for XOR-CIM hardware performance and protection overhead. The XOR-CIM system-level hardware performance is evaluated using a modified NeuroSim [20] with the XOR-CIM subarray calibrated in SPICE using a 28nm foundry process. 16-bit inputs & 16-bit weights for the first and last layer and 8-bit inputs & 8-bit weights for the rest layers are assumed for the networks to guarantee no accuracy loss after quantization. For the ImageNet classification case, 4-bit and 5-bit ADC resolutions are used for XOR-CIM and Normal-CIM settings, respectively. At the same time, the precision decreases to 2-bit and 3-bit ADC for CIFAR-10 classification. Inside the XOR-CIM core, several 64×64 subarrays constitute the processing element (PE) according to the weight precision. At the subarray level, the multi-bit activations are fed sequentially with outputs accumulated with shift & add circuits to get VMMs of high-precision input and 1-bit weight. Another shift-add on the PE level accumulates the outputs of different weight subarrays to get the final high-precision VMM result. Considering the broad use of 3×3 kernel size, a group of 9 PEs forms a tile so that most layers could fit in exactly multiple tiles [70]. Adder trees accomplish further addition among tiles, and an H-tree undertakes communication. After each CONV/FC layer, pooling and/or activations are done by digital modules.

5.4.2 Hardware performance of the XOR-CIM core

A benchmark of our proposed XOR-CIM architecture with three baselines is shown in Table 5. The Normal-CIM uses the conventional 6T SRAM to do CIM operations without encryption. The Normal-CIM (Dual-WL) mode means the array consists of Dual-WL 6T SRAM cells, but only one side is used for conventional CIM operation. Still, no encryption is supported in this mode. The near-memory calculation reads the encrypted weight, decrypts it, and conducts the product-and-sum using digital circuits. For all three evaluated networks, the throughputs of all the CIM cases are similar, while 3~4 times better than the near-memory case due to the high parallelism of the CIM operation. Compared to the 6T SRAM array, the Dual-WL 6T SRAM is more area-consuming, increasing the total area for both XOR-CIM and Normal-CIM (Dual-WL). Also, the circuits for encryption are not included for both Normal-CIM cases, introducing an area overhead to XOR-CIM.

However, the increment is still acceptable, limited to 2.5%~11.25% based on the network structure. Nevertheless, the near-memory case has the largest area due to the additional MAC units. Interestingly, the Dual-WL 6T SRAM is more energy-efficient than the normal 6T case, improving the XOR-CIM by 22%~56%. We think the reason is that the Normal-CIM operation will open both access transistors during calculation, causing a waste of energy on $\overline{\mathrm{BL}}$. This assumption has been approved by comparing the Normal-CIM (Dual-WL)'s and Normal-CIM's energy efficiency results.

Table 5 - Benchmark hardware performance among Normal-CIM, near-memory compute, and XOR-CIM.

Technode: 28nm					Precision 8-bits
		Normal-CIM	Normal-CIM (Dual-WL)	Near Memory	XOR-CIM
ResNet-18	TOPS/W	2.03	3.23	1.54	2.86
	GOPS	770	770	236	770
	Area(mm^2)	200	226	296	205
VGG-11	TOPS/W	1.55	2.04	0.7	1.9
	GOPS	1,212	1,212	314	1,212
	Area(mm^2)	1,638	1,730	2,172	1,692
VGG-8	TOPS/W	3.53	6.92	2.36	5.53
	GOPS	2,985	2,985	943	2,985
	Area(mm^2)	160	169	257	178

5.4.3 Overhead for secure communication

The hardware costs of the XOR-CIM core, SHA256, DRBG, GEN, and AES are shown in Table 6. Taking ResNet-18 as an example, the breakdowns for latency, energy, and area are shown in Figure 39, considering the whole network is encrypted while only one inference is running in the XOR-CIM core. The results show that, for encryption and authentication, the energy and area cost is relatively small while the processing time occupies almost 38%. Thus, the latency overhead becomes the bottleneck of our secure inference engine.

Table 6 - Hardware performance of on-chip blocks at 28nm for implementing the entire ResNet-18 secure inference engine.

	CIM core	SHA256 [40]	DRBG [40]	GEN [17]	AES [41]
Area (mm^2)	205	0.0197	0.0575	0.0014	0.0028
Power (mW)	269.2	11.12	10.00	0.654	0.45
Latency (ns)	1.12E6	43.86	2.78E4	1.33	7.65

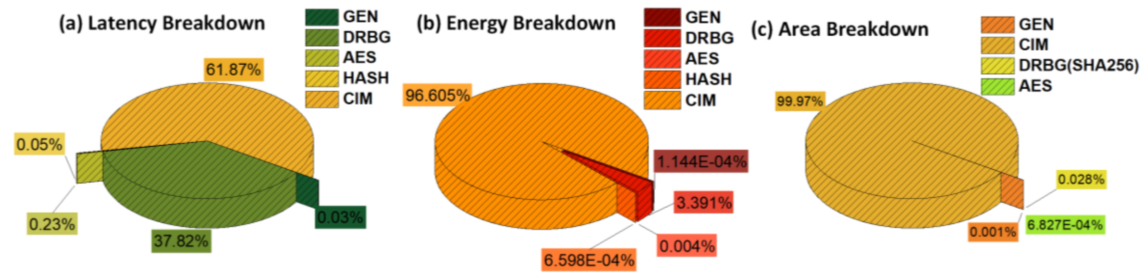


Figure 39 - Latency, energy and area breakdown for the secure inference engine.

However, the assumption we make here is relatively aggressive. On one side, it is impractical to run only one inference for each download. Thus, this setup time consumption could be amortized by inferencing more images. On the other side, encrypting the DNN model for protection is not necessary. We can look for a balance point in practice between hardware overhead and security level. By exploring the impact on inference accuracy, we find that only a single layer's encryption could greatly degrade the accuracy, as shown in Figure 40 (a)-(f). We further reduce the encrypted fragment inside a kernel window and find that, to some layers, 6/9 of the layer being encrypted is enough to ruin the functionality of the network Figure 40 (g)-(l).

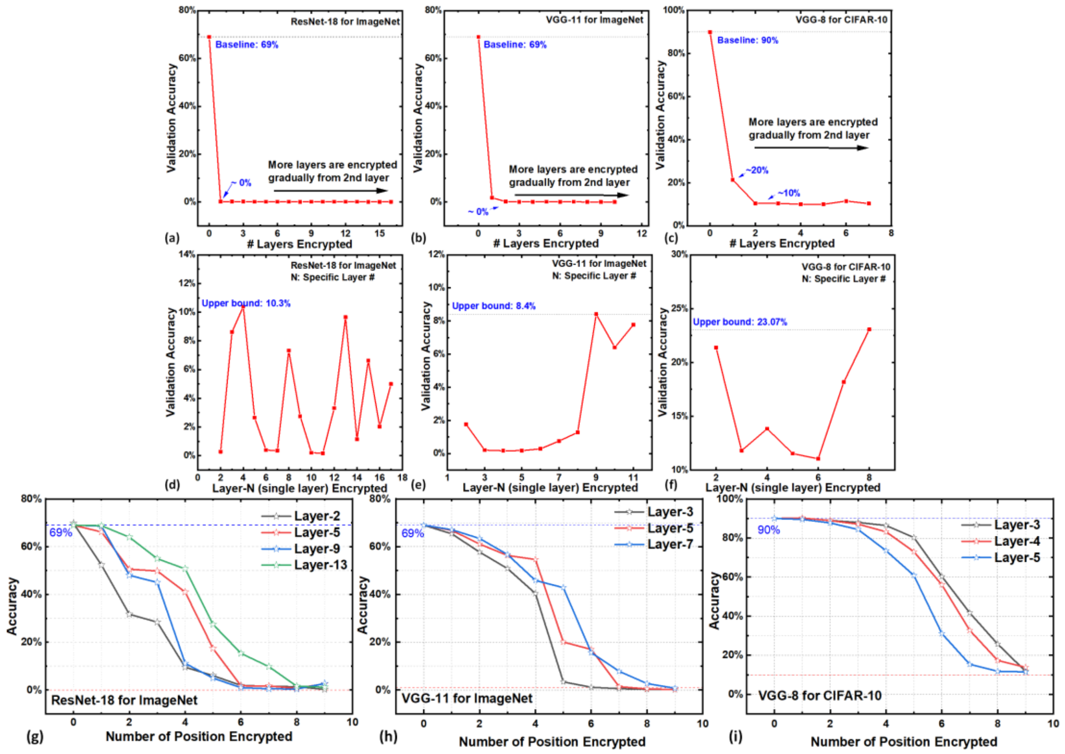


Figure 40 - Inference accuracy vs. portion of network encrypted.

5.4.4 Vulnerability of XOR key bit sharing

As aforementioned, the CIM operation naturally forces the weight in the same row to share a key bit, which increases the vulnerability of being cracked by frequency analysis. As shown in Figure 41, the natural weight distribution of a layer is zero-centered. However, XOR encryption will make the weights symmetry along the dashed line. In other words, there should originally be more digit 0 in the raw weights. The XOR encryption increased the number of digit 1 by flipping the weight digits seeing a key bit 1, making the encrypted model have an equal portion of digits 0 and 1. Thus, the distribution of the weight bits encrypted by the same key could imply if the key bit is 1 or 0.

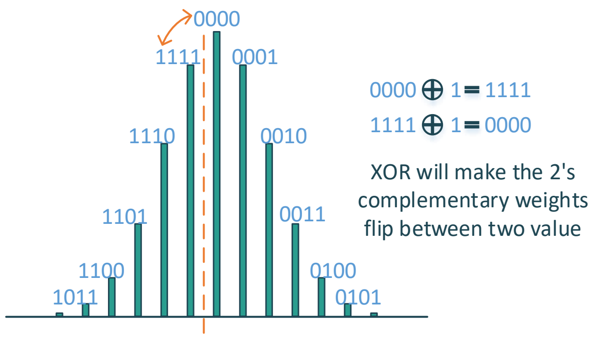


Figure 41 - Illustration of weight encryption for 4-bit weight.

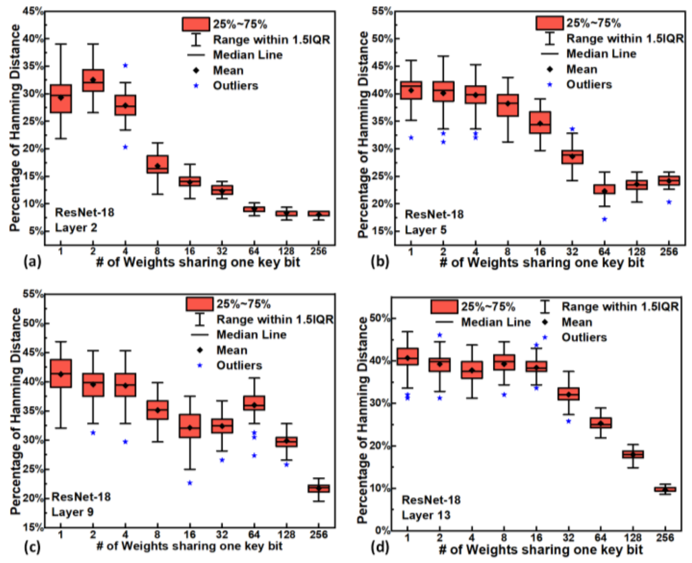


Figure 42 - The hamming distance between the predefined key and the key found by frequency analysis with several weights sharing one key.

Figure 42 shows the percentage of the key incorrectly restored using the method above (Hamming distance between the real key and the restored key over the key length). We could observe that the more weights share the key, the easier it is to infer it. However, while it is desired to have all columns work simultaneously for a CIM array, it is rare to manage it in the real chip due to the pitch mismatch between the column and the periphery circuits. In our 64x64 subarray setting, every eight columns are assumed to share one set of periphery circuits. Therefore, only eight weights in the same row have to share one key bit instead of 64. From the result, sharing 8 bits could still maintain around a 35%

difference between the real and restored keys for some layers. This result could be combined with the previous results of partial encryption to analyze the necessary portion of the encryption. Applying a random key to attack the XOR encryption will lead to ~50% bits match. If 6/9 of the layer encrypted is enough to protect the network, it means 33% mismatch could maintain the low accuracy of the encrypted network. The prior results are evaluated with no key bit sharing. Considering that a key bit shared among 8 bits will reduce the 50% mismatch to 35%, at least one layer needs to be encrypted.

5.5 Conclusion

Our research proposes an SRAM-based XOR-CIM Inference Engine with a protocol for secure communication with cloud databases as a two-party system. The protocol is modified from the reverse secure sketch protocol for key processing of the XOR-based stream cipher and chip authentication. Correspondingly, the accelerator architecture is presented with function module sharing to reduce overhead. As the core of the inference engine, XOR-CIM is proposed to support CIM operation directly on XOR encrypted weight, with similar throughput, small area overhead, and better energy efficiency than the conventional 6T SRAM-based CIM accelerator. Finally, the overhead and risk introduced by the proposed inference engine are analyzed to prove its feasibility.

CHAPTER 6. Secure eNVM-CIM Engine: Exploiting Process Variations to Protect Machine Learning Inference Engine from Chip Cloning and Adversarial Attack

6.1 Motivation

Unlike the SRAM-based CIM engine, which needs to download the DNN model for each power-on, the eNVM-based CIM could always hold the weights on-chip. While this non-volatility is attractive to the edge DNN accelerator, eNVM subarrays suffer from the same information leakage problem as those used for memory-only purposes.

We integrate XOR-cipher into the array for the SRAM-based CIM architecture as the weight and inversed weight naturally exist on-chip. For the eNVM-based CIM architecture, weights are generally mapped to the cell's conductance. Thus, there is no straightforward way to perform analog computations on digitally encrypted weights as in the SRAM case. In this work, we explore the vulnerabilities of eNVM-based CIM chips and propose a lightweight countermeasure to defend against them.

6.2 Vulnerability of eNVM-based CIM chip

6.2.1 Chip cloning attack

Data privacy is generally a problem for eNVMs when raw data is stored on-chip without encryption [6]. This problem will be severer for edge devices as they are not always used in a secure environment. Thus, for the CIM inference engine with memory cells holding raw weights of DNN models, there will be potential threats of chip cloning.

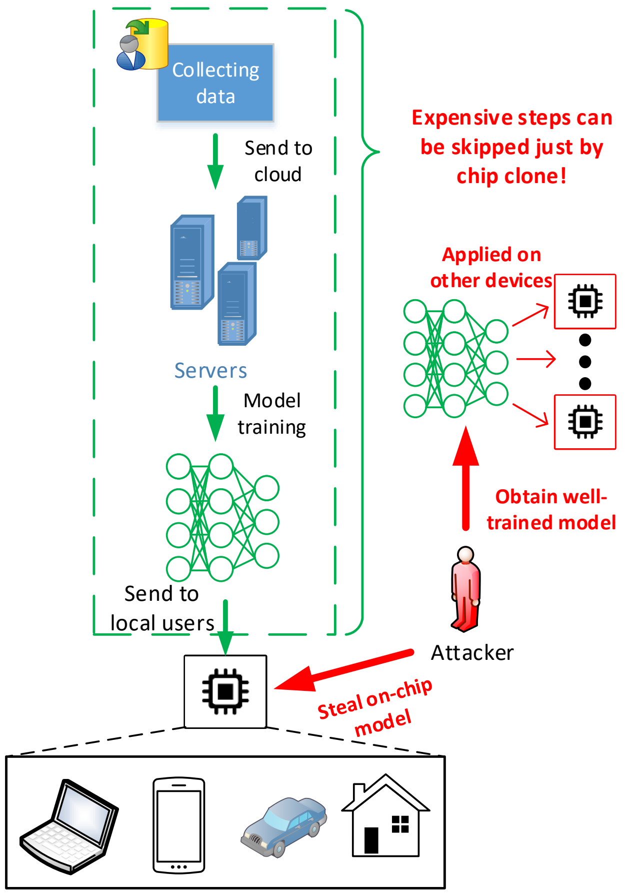


Figure 43 - Chip-cloning attack that bypass the expensive process of data/label collection and model training.

As aforementioned, the DNN model stored in the eNVM-based inference engine could be the business property of the model owner, considering the substantial efforts of gathering the training data/label and training the network. If the adversary can get an eNVM-based device, he/she could read out the model's weights as they are saved in raw format in the non-volatile cells. Then the adversary could reprogram (clone) the weights to another chip without going through the expensive data gathering and model training procedure, which is defined as the chip cloning attack in this work (Figure 43).

In theory, this goal can be achieved by the micro-probing attack. According to [7], the high density of eNVM cell array may make it challenging to directly probe a single cell without physically damaging the neighboring cells. Alternatively, the adversary could directly read out the weights row by row by probing the digital output from the periphery, e.g. the analog-to-digital converter (ADC). Normally, the ADC used for the partial sum computation does not require super high resolution [8] and thus will have information loss for row-by-row reading. However, due to the non-idealities of the eNVM cell writing, a high-resolution ADC is usually required in eNVM-based CIM for the write-verify scheme to minimize the cell conductance variations when loading the DNN models on-chip. This ADC could become a vulnerable spot later.

6.2.2 Adversarial attack

Besides the model's weights, the leaked model could cause some security problems considering the functionality. While deep neural networks (DNNs) have achieved outstanding progress in various applications, there is a growing concern regarding adversarial attacks, which aim to fool the model with manipulated inputs [71] while not affecting human decisions. Currently, adversarial attacks are generally grouped into two categories considering the information of the target model exposed to the adversary. If the adversary has full access to the DNN model architecture and weights, it is called the white-box attack [72, 73]. If the adversary has only external access to the network (e.g., input and output), it will be categorized as a black-box attack. The white-box attack could achieve higher success rates than the black-box attack [72], while it is rarer to happen in real life. Generally, it is not easy for adversaries to access a private model in the data center or the cloud. However, regarding the eNVM-based CIM inference engine, the edge devices are physically accessible by anyone and thus could leak the model information at high risk. Extra algorithmic calculations could be used to defend against the white-box attack at the expense of speed and power overhead, which is undesired for the edge device with a limited power budget and demanding a real-time response. Instead of on-chip defending against the adversarial attack, this work is concerned with the transferability of the adversarial examples among chips loading the same model. Normally, the chips are designed for mass use, which means chips with the same structure and loaded with the same model will be widely equipped in different edge devices. Thus, adversaries only need to get the DNN model from a certain chip, and then they could generate adversarial examples that could fool the rest of the chips. As a result, all the chips could be disabled by a single weak point as shown in Figure 44.

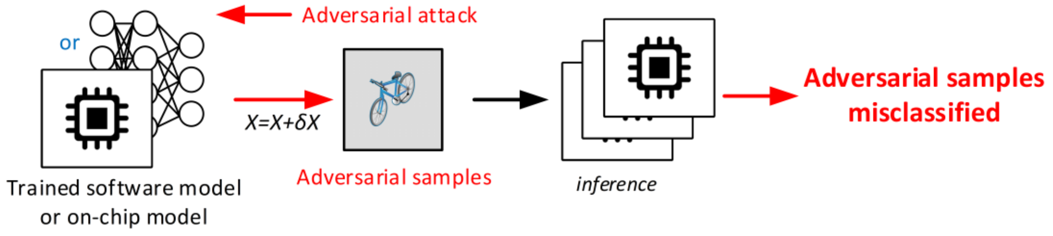

or
Adversarial attack
$X=X+\delta X$
Adversarial samples
inference
Adversarial samples misclassified
Trained software model or on-chip model

Figure 44 - The treat of generating adversarial examples from one chip and affecting all the rest chips.

6.3 Security benefit

6.3.1 Protect machine learning inference engine from chip cloning

As illustrated in Chapter 3, we know that the ADC offset caused by process variation will degrade the model accuracy on-chip and can be compensated by on-chip fine-tuning. After fine-tuning, the model weights on-chip will be different from the original model and other chips because of the different ADC offset patterns. Inspired by PUF, We could utilize this fine-tuning under ADC offset to protect the model on-chip against chip cloning. In more detail, we view the fine-tuning of the model on-chip as a process to fit the model exactly against each chip's unique ADC offset pattern. As a result, the model only works under a certain chip with the specific ADC offset pattern. Since the ADC offset caused by the process variation is unclonable, the model on-chip will also become unclonable. However, due to the noise robustness of DNNs, they may still work under different offset patterns. Thus, the ADC structure adopted on-chip must satisfy the following three requirements to defend against the chip cloning attack. First, the ADC offset should be big enough to cause an obvious accuracy loss on the original model. Second, the model on-chip should be able to fully recover the accuracy after fine-tuning. Finally, when the fine-tuned model is cloned to other chips, it should have a bad performance so that the cloning could be identified as failed. Referring to Figure 17, only SAR-ADC with certain transistor sizes can achieve the first two conditions and be used as candidates for the chip-cloning defense. Then we need to check if these settings could satisfy the final requirement.

According to Figure 17, SAR-ADCs with W/L=4 for 2-bit weights and W/L=3 for 4/8-bit weights are candidates to defend against the chip cloning attack. First, more chip samples with specified ADCs are fine-tuned to check the generality of accuracy recovery under these settings, as shown in Figure 45 (a). More than 90%, 89%, and 87% accuracy could be achieved for these three cases separately across several retaining tests. Then, the weight cloning attack is assumed on these fine-tuned chips, which is to read out the weights on-chip by the probing attack and apply them to other fake chips with the same settings but different ADC offset patterns. Figure 44 (b) shows that the retrained models have a relatively low accuracy of 20%~40% on other chips. It is worth noticing that this fine-tuning differs from fine-tuning the network to be more noise-robust by injecting noise. The noise pattern caused by ADC offset is static instead of stochastic during training. The fine-tuned model is overfitted to the specific pattern instead of becoming more noise-robust. Thus different ADC offset patterns will cause an accuracy drop again.

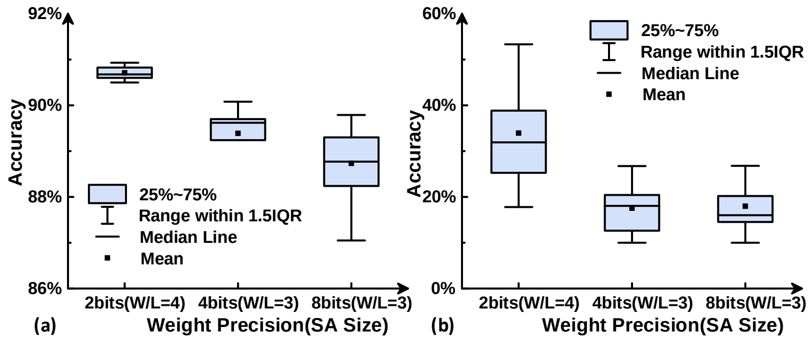


Figure 45 - Inference accuracy distribution of software-trained model with ADC offset.

6.3.2 *Mitigating transferability of adversarial examples*

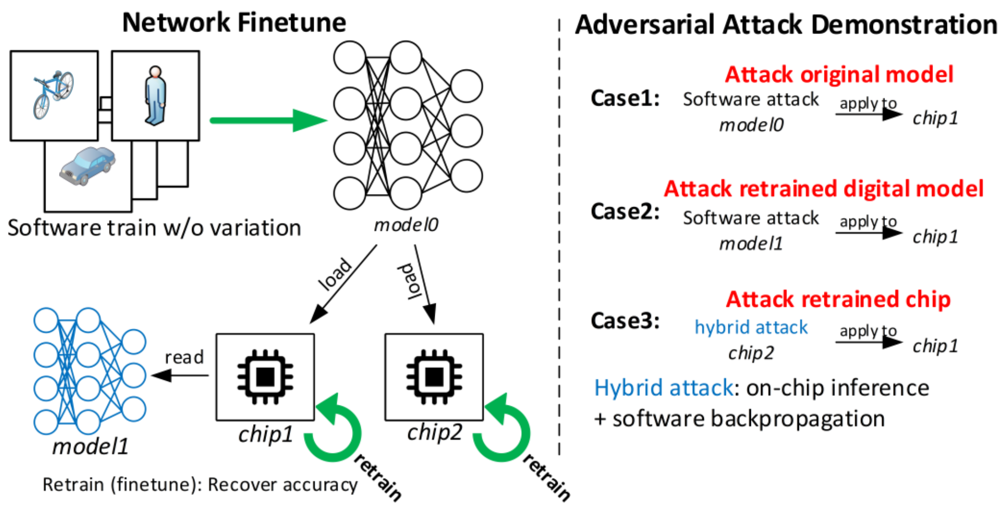


Figure 46 - Three adversarial attack scenarios: Case1: attack the original model in software and apply the examples on a fine-tuned chip; Case2: attack the digital model read out from a fine-tuned chip and apply the generated examples on a fine-tuned chip; Case3: attack the model on-chip directly in a hybrid way and apply the generated examples on another fine-tuned chip.

The white-box adversarial attack is more efficient than the black-box attack as it utilizes the network weights to generate adversarial examples. Correspondingly, these adversarial examples are more closely related to the model's weights. To compensate for the accuracy loss caused by ADC offset, the weights will be chip-wisely different by fine-tuning. This chip-to-chip variation could bring us a byproduct: the chip will be robust to the adversarial examples transferred from other chips or software baselines. Explicitly, even if the adversary gets the software model or a certain chip and attacks the model/chip instance to generate some adversarial examples, he/she could not use the same examples to fool other chip instances due to the uniqueness of the DNN model on each chip.

We evaluate the proposed transferability mitigation method with VGG-8 and DenseNet-40 networks for the CIFAR-10 dataset. The VGG-8 model has 8-bit activations and 2-bit weights, while 8-bit activation and 8-bit weight are adopted for DenseNet-40. Both networks could achieve ~92% accuracy as software baseline on CIFAR-10 classification. We evaluate a white-box attack called Carlini and Wagner (C&W) Attack [73] on these two models assuming three different scenarios as shown in Figure 46: Attack original model; Attack retrained digital model; Attack retrained chip. Here, model0 means the original model trained in software without chip variations and will be loaded to all the chips as an initial condition. The chips will be fine-tuned to recover the accuracy degradation caused by ADC offset. Considering a certain fine-tuned chip1, if the model on chip1 is read out, it will be different from model0 because of fine-tuning, and we refer to this new model as model1. Finally, we use chip2 to denote a fine-tuned chip with a distinct ADC offset pattern from chip1.

Table 7 presents accuracy results after fine-tuning and the performance under three attack cases with different ADC settings. As shown in the table, chip accuracy could be generally recovered to baseline accuracy (above 90%) by retraining for all cases. Unlike the chip-cloning defense case, the big accuracy drop is unnecessary for this adversarial examples transferability mitigation purpose. Thus, there are no strict requirements on ADC except for fine-tuning to recover the accuracy loss caused by ADC offset. In the case1 scenario, the adversarial attack is applied to the original software baseline model (model0). It can be seen that the attack is quite effective and degrade the classification accuracy to ~0%. However, when applying the generated adversarial examples on chip1, the retrained network on-chip can still preserve relatively high accuracy (~75% for VGG-8, ~84% for DenseNet-40) on these manipulated inputs.

Table 7 - Accuracy performance under C&W attack ($\boldsymbol{L_2}$).

	Chip Information			**Attack original model**	
Chip config.	ADC type	W/L	Retrained accuracy	Software Attack(model0)	Attack on chip1
VGG-8					
A	SAR	9	89.39%	0.61%	73.95%
B	SAR	10	90.87%		75.12%
C	Flash	9	91.36%		74.10%
D	Flash	10	91.46%		74.40%
DenseNet-40(k=24)					
A	SAR	9	91.04%	0%	84.59%
B	SAR	10	91.52%		83.11%
C	Flash	9	91.50%		85.56%
D	Flash	10	91.81%		84.19%
	Attack retrained digital model			**Attack retrained chip**	
Chip config.	Digital accuracy (model1)	Software Attack(model1)	Attack On chip1	Chip2 acc. after attack	Attack on chip1
VGG-8					
A	74.75%	0.09%	83.43%	0%	62.10%
B	83.89%	0.24%	78.78%		64.80%
C	89.31%	0.15%	65.73%		65.10%
D	90.54%	0.21%	51.22%		64.30%
DenseNet-40(k=24)					
A	20.04%	0%	87.69%	0%	87.20%
B	35.25%	0%	89.52%		85.25%
C	62.71%	0%	87.62%		86.80%
D	85.07%	0%	84.65%		86.30%

For attack case 2, the adversary is assumed to have access to a fine-tuned chip (chip1) but could only conduct the adversarial attack in software. Thus, he/she first reads out the retrained model (model1), which has been fine-tuned to fit the ADC variation on chip1 and then uses it to generate adversarial examples. There will be an accuracy degradation for the model1 since it ignores the contribution of ADC offset on-chip. Its digital accuracy can be further decreased to ~0% by the C&W attack. However, while applying these pure software-generated adversarial examples back to chip1, it can maintain relatively high accuracy thanks to the ADC variation. In attack case 3, we assume the adversary could generate adversarial examples from the model on-chip (chip2) directly using a hybrid way. The inference is performed on chip2, and a software platform conducts backpropagation. This hybrid attack is similar to hybrid fine-tuning. Under such a scenario, the chip under attack will be disabled with 0% accuracy, but other fine-tuned chips (chip1) could maintain a certain accuracy.

Table 8 - C&W attack on VGG-8 with different distance matrices.

		Attack original model		**Attack retrained chip**	
		Software attack	Attack on chip1	Chip2 acc. after attack	Attack on chip1
Acc. before attack		91.96%	88.10%	90.50%	90.78%
C&W attack	$\boldsymbol{L_0}$ attack	0.57%	73.54%	0.26%	71.30%
	$\boldsymbol{L_2}$ attack	0.68%	74.23%	0.02%	63.40%
	$\boldsymbol{L_\infty}$ attack	2.61%	73.35%	0.88%	70.10%

We further vary the distance matrices used in the C&W attack on VGG-8 ($\boldsymbol{L_0}, \boldsymbol{L_2}, \boldsymbol{L_\infty}$) and, as shown in Table 8, the proposed defense is effective regardless of the used norm type.

6.4 Summary

This chapter identifies two threats to eNVM-based machine learning inference engines: the chip cloning attack and the transferability of adversarial examples. We propose a PUF-like scheme that comes free with the on-chip fine-tuning to recover the accuracy loss caused by the process variation. On one side, the fine-tuned model could maintain high accuracy on each chip instance, while its performance will significantly degrade on other chip instances with cloned weights. Thus, the threat of chip-cloning is released. On the other side, accompanied by accuracy recovery, updated weights on-chip will vary from chip to chip. As a result, the transferability of the adversarial examples is strongly suppressed by fine-tuning. While the classification accuracy of the original attacked chip/models drops to almost 0%, the software performance of other chips could still be maintained to some extent.

CHAPTER 7. Conclusion

7.1 Key contribution

This thesis is focused on the reliability and security issues of CIM accelerators. These two aspects are equally important for using the CIM accelerators as edge devices practically. They could also be combined so that protections come free with the techniques to improve reliability, reducing the implementation overhead. The contributions of this thesis include:

We define the basic design flow of algorithm-to-hardware mapping of CIM and explore its design space. Two quantization methods with three mapping approaches are demonstrated for hardware performance evaluation. Detailed analysis of these results provides a deeper insight into the reliable and hardware-friendly system-level CIM design. While this analysis is done for CIM, some common ideas could be expanded to accelerators in other domains, such as purely digital architectures. First, as some digital accelerators adopt zero-skip schemes to reduce power consumption, quantization methods with higher input sparsity could also help decrease their real computation energy efficiency. Second, a similar bit-wise MAC could be implemented in the digital domain with AND gate and adder-tree, followed by the shift-adders. All the number representation schemes could be applied to it with binary digits. This way, a potential precision reduction could be applied to the adder-trees based on the ADC results. Finally, the tolerance of ADC precision loss implies that besides quantizing the inputs and weights, further precision loss during product accumulation is possible even under the traditional MAC operation in digital, which could introduce hardware overhead reduction on the accumulator.

After that, we introduce process variations into the CIM inference engine, which will cause ADC offset and thus hurt the software performance. To recover the inference accuracy, we proposed a hybrid on-chip fine-tuning method. It is shown that the on-chip fine-tuning is beneficial to compensate for the ADC offset effects. Other works also show that training with noise injected to mimic the effect of memory cell variations and IR drops could also recover the DNN performance on-chip. As a result, we propose to train the network directly on the chip. We find that the non-idealities of the devices and circuits will degrade the training accuracy, and the DRAM access will hurt its energy efficiency. Momentum is utilized during training to overcome the performance degradation caused by non-idealities, and we propose a segmented update scheme to reduce the DRAM access. This performance recovery is specific to the analog operations since the digital system normally has a good noise margin for signal recovery and thus faces much fewer non-ideal effects. Also, it takes advantage of the DNN’s noise tolerance and self-adaption properties to obtain good performance under non-idealities. However, the segmented gradient calculation and weight update scheme could be applied to any system to reduce the DRAM access for training.

To protect the raw on-chip weights in CIM inference designs, we first aim to develop a secure SRAM-based XOR-CIM engine with a modified reverse secure sketch protocol to enable on-chip authentication and key processing for XOR-based stream cipher encrypted models. The evaluation results show that the proposed XOR-CIM could enhance security, achieving comparable energy efficiency and no throughput loss, with negligible area overhead compared to conventional CIM designs with no protection. The concept of using a complex encryption method to encrypt the seed while using a relatively simple one to encrypt the big model could be applied to any edge device to reduce the encryption overhead. In addition, this XOR-cipher integrated SRAM can also be used for memory purposes to hold encrypted data on-chip. It can save the extra operation for standalone XOR decryption of the data. However, the energy improvement from Dual-WL SRAM cells for memory purposes may not be big because of the single-row operation.

Unlike the SRAM-based CIM engine, it is hard to integrate encryption into the eNVM-based CIM engine. Inspired by the necessary retraining to recovery accuracy under process variation, we propose a PUF-like scheme against the weight cloning attack and mitigate the transferability of the adversarial example. The individual chip can maintain high accuracy after fine-tuning, while its performance will significantly degrade on other chip instances with cloned weights. The fine-tuned chips also become more robust to the adversarial examples generated from the pure software model or different chips. While we utilize the ADC offset pattern to provide the PUF-like specification of the model, any other unclonable source from hardware could be integrated into the model and achieve the same purpose as the ADC offset. As a result, this protection is not limited to the CIM but is possible for all DNN accelerators.

7.2 Future work

While CIM accelerators have shown superior performance and efficiency over conventional digital computing systems, the transition from von Neumann architectures to in-memory computing platforms involves cross-layer considerations from hardware to software. We believe further efforts need to be spent in several directions to enable the widespread use of CIM.

Firstly, considering the immaturity of the eNVMs technologies, CIM architectures necessitate an integrated mapping of various non-idealities to explore software/hardware co-designs that can deal with devices' unavoidable reliability issues. While different works have explored parts of non-idealities in their work, it is desired to have a flexible and all-sided simulator to cover different cases of non-idealities for better early-stage estimation of CIM design.

Secondly, further exploration of the security vulnerabilities and countermeasures in CIM is desired to allow the in-memory computing solution to be used in real-world computing systems. Also, these methods, including the solutions proposed in this work, should not be validated only in simulation but also with real chip implementation.

Finally, system-level CIM should be evaluated for big-scale tasks. Currently, most of the CIM is demonstrated for small networks. For complicated tasks, the performance of both software/hardware is evaluated in a hybrid way, which integrates the macro-level silicon data into the simulator to predict the system-level result. To move CIM from the research community to industry, it is important to validate its strength in silicon with real-life tasks. Thus, validating a big-scale system-level CIM prototype chip is one of the most import future tasks for researchers in this area.

REFERENCES

[1] Dave Steinkrau, Patrice Y. Simard, Ian Buck, "Using GPUs for Machine Learning Algorithms," in *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, 2005.

[2] Zhao Wang, Yijin Guan, Guangyu Sun, Dimin Niu, Yuhao Wang, Hongzhong Zheng, Yinhe Han, "GNN-PIM: A Processing-in-Memory Architecture for Graph Neural Networks," in *Conference on Advanced Computer Architecture*, 2020.

[3] Jin Luo, Weikai Xu, Yide Du, Boyi Fu, Jiahao Song, Zhiyuan Fu, Mengxuan Yang, Yiqing Li, Le Ye, Qianqian Huang, Ru Huang, "Energy- and Area-efficient Fe-FinFET-based Time-Domain Mixed-Signal Computing In Memory for Edge Machine Learning," in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, 2021.

[4] Han Zhao, Zhengwu Liu, Jianshi Tang, Bin Gao, Ying Zhou, Peng Yao, Yue Xi, He Qian, Huaqiang Wu, "Implementation of Discrete Fourier Transform using RRAM Arrays with Quasi-Analog Mapping for High-Fidelity Medical Image Reconstruction," in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, 2021.

[5] Ming-Chun Hong, Le-Chih Cho, Chih-Sheng Lin, Yu-Hui Lin, Po-An Chen, I-Ting Wang, Pei-Jer Tzeng, Shyh-Shyuan Sheu, Wei-Chung Lo, Chih-I Wu, Tuo-Hung Hou, "In-Memory Annealing Unit (IMAU): Energy-Efficient (2000 TOPS/W) Combinatorial Optimizer for Solving Travelling Salesman Problem," in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, 2021.

[6] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, Vivek Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, Seoul, 2016.

[7] Shibin Tang, Shouyi Yin, Shixuan Zheng, Peng Ouyang, Fengbin Tu, Leiyue Yao, JinZhou Wu, Wenming Cheng, Leibo Liu, Shaojun Wei, "AEPE: An area and power

efficient RRAM crossbar-based accelerator for deep CNNs," in *IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, Hsinchu, 2017.

[8] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Annual Conference on Neural Information Processing Systems*, Lake Tahoe, 2012.

[9] Karen Simonyan, Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations,* San Diego, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, 2016.

[11] Xiaoyu Sun, Shihui Yin, Xiaochen Peng, Rui Liu, Jae-sun Seo, Shimeng Yu, "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2018.

[12] Shihui Yin, Xiaoyu Sun, Shimeng Yu, Jae-Sun Seo, "High-Throughput In-Memory Computing for Binary Deep Neural Networks With Monolithically Integrated RRAM and 90-nm CMOS," *IEEE Transactions on Electron Devices,* vol. 67, no. 10, pp. 4185 - 4192, 2020.

[13] Cheng-Xin Xue, Wei-Hao Chen, Je-Syu Liu, Jia-Fang Li, Wei-Yu Lin, et al., " 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6ns Parallel MAC Computing Time for CNN Based AI Edge Processors," in *IEEE International Solid-State Circuits Conference - (ISSCC),* San Francisco, 2019.

[14] Cheng-Xin Xue, Tsung-Yuan Huang, Je-Syu Liu, Ting-Wei Chang, et al., "A 22nm 2Mb ReRAM Compute-in-Memory Macro with 121-28TOPS/W for Multibit MAC Computing for Tiny AI Edge Devices," in *IEEE International Solid- State Circuits Conference - (ISSCC),* San Francisco, 2020.

[15] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Tingwen Huang, Qing Wu, Mark Barnell, "Reduction and IR-drop compensations techniques for reliable neuromorphic

computing systems," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, 2014.

[16] Zhezhi He, Jie Lin, Rickard Ewetz, Jiann-Shiun Yuan, Deliang Fan, "Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping," in *IEEE/ACM Design Automation Conference (DAC)*, New York, 2019.

[17] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, Mingoo Seok, "XNOR-SRAM: In-Memory Computing SRAM Macro for Binary/Ternary Deep Neural Networks," *IEEE Journal of Solid-State Circuits,* vol. 55, no. 6, pp. 1733 - 1743, 2020.

[18] Jong-Hyeok Yoon, Muya Chang, Win-San Khwa, Yu-Der Chih, Meng-Fan Chang, Arijit Raychowdhury, "A 40nm 64Kb 56.67TOPS/W Read-Disturb-Tolerant Compute-in-Memory/Digital RRAM Macro with Active-Feedback-Based Read and In-Situ Write Verification," in *IEEE International Solid- State Circuits* Conference *(ISSCC)*, San Francisco, 2021.

[19] Avishek Biswas, Anantha P. Chandrakasa, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," in *IEEE International Solid - State Circuits Conference - (ISSCC)*, San Francisco, 2018.

[20] Xin Si, Jia-Jing Chen, Yung-Ning Tu, Wei-Hsing Huang, et al., "Twin-8T SRAM Computation-In-Memory Macro for Multiple-Bit CNN-Based Machine Learning," in *IEEE International Solid- State Circuits Conference - (ISSCC)*, San Francisco, 2019.

[21] Akhilesh Jaiswal, Indranil Chakraborty, Amogh Agrawal, and Kaushik Roy, "8T SRAM Cell as a Multibit Dot-Product Engine for Beyond Von Neumann Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 27, no. 11, pp. 2556 - 2567, 2019.

[22] Wonbo Shim, Yandong Luo, Jae-sun Seo, Shimeng Yu, "Impact of Read Disturb on Multilevel RRAM based Inference Engine: Experiments and Model Prediction," in *2020 IEEE International Reliability Physics Symposium (IRPS)*, Dallas, 2020.

[23] Wei-Hao Chen, Chunmeng Dou, Kai-Xiang Li, Wei-Yu Lin, et al., "CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors," *Nature Electronics,* vol. 2, p. 420–428 , 2019.

[24] Bing Li, Bonan Yan, Hai Li, "An Overview of In-memory Processing with Emerging Non-volatile Memory for Data-intensive Applications," in *Great Lakes Symposium on VLSI* , Tysons Corner, 2019.

[25] Wei-Hao Chen, Kai-Xiang Li, Wei-Yu Lin, Kuo-Hsiang Hsu, Pin-Yi Li, et al., "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *IEEE International Solid - State Circuits Conference - (ISSCC),* San Francisco, 2018.

[26] Yandong Luo, Xiaochen Peng, Ryan Hatcher, Titash Rakshit, Jorge Kittl, Mark S. Rodder, Jae-Sun Seo, Shimeng Yu, "A Variation Robust Inference Engine Based on STT-MRAM with Parallel Read-Out," in *IEEE International Symposium on Circuits and Systems (ISCAS),* Seville, 2020.

[27] Wangxin He, Shihui Yin, Yulhwa Kim, Xiaoyu Sun, Jae-Joon Kim, Shimeng Yu, Jae-Sun Seo, "2-Bit-Per-Cell RRAM-Based In-Memory Computing for Area-/Energy-Efficient Deep Learning," *IEEE Solid-State Circuits Letters,* vol. 3, pp. 194 - 197, 2020.

[28] Pai-Yu Chen, Xiaochen Peng, 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,* vol. 37, no. 12, pp. 3067 - 3080, 2018.

[29] Xiaoyu Sun, Shimeng Yu, "Impact of Non-Ideal Characteristics of Resistive Synaptic Devices on Implementing Convolutional Neural Networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* vol. 9, no. 3, pp. 570 - 579, 2019.

[30] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M. Shelby, Irem Boybat, Carmelo di Nolfo, et al, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature,* vol. 558, p. 60–67, 2018.

[31] Xiaoyu Sun, Panni Wang, Kai Ni, Suman Datta, Shimeng Yu, "Exploiting Hybrid Precision for Training and Inference: A 2T-1FeFET Based Analog Synaptic Weight Cell," in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, 2018.

[32] Tayfun Gokmen, Wilfried Haensch, "Algorithm for training neural networks on resistive device arrays," *Frontiers in neuroscience*, vol. 14, no. 103, 2020.

[33] Chaofei Yang, Beiye Liu, Hai Li, Yiran Chen, Mark Barnell, Qing Wu, Wujie Wen, Jeyavijayan Rajendran, "Thwarting Replication Attack Against Memristor-Based Neuromorphic Computing System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* , vol. 39, no. 10, pp. 2192 - 2205, 2020.

[34] Yi Cai, Xiaoming Chen, Lu Tian, Yu Wang, Huazhong Yang, "Enabling Secure in-Memory Neural Network Computing by Sparse Fast Gradient Encryption," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Westminster, 2019.

[35] Shanshi Huang, Hongwu Jiang, Shimeng Yu, "Hardware-aware Quantization/Mapping Strategies for Compute-in-Memory Accelerators," *ACM Transactions on Design Automation of Electronic Systems*, accepted, 2022.

[36] Shanshi Huang, Xiaochen Peng, Hongwu Jiang, Yandong Luo, Shimeng Yu, "Exploiting Process Variations to Protect Machine Learning Inference Engine from Chip Cloning," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021.

[37] Shanshi Huang, Xiaoyu Sun, Xiaochen Peng, Hongwu Jiang, Shimeng Yu, "Overcoming Challenges for Achieving High in-situ Training Accuracy with Emerging Memories," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, 2020.

[38] Shanshi Huang, Xiaoyu Sun, Xiaochen Peng, Hongwu Jiang, Shimeng Yu., "Achieving High In Situ Training Accuracy and Energy Efficiency with Analog Non-Volatile Synaptic Devices," *ACM Transactions on Design Automation of Electronic Systems*, vol. 27, no. 4, pp. 1-19, 2022.

[39] Shanshi Huang, Hongwu Jiang, Xiaochen Peng, Wantong Li, Shimeng Yu, "XOR-CIM: Compute-In-Memory SRAM Architecture with Embedded XOR Encryption,"

in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, San Diego, 2020.

[40] Shanshi Huang, Hongwu Jiang, Xiaochen Peng, Wantong Li, Shimeng Yu, "Secure XOR-CIM Engine: Compute-In-Memory SRAM Architecture With Embedded XOR Encryption," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 12, pp. 2027-2039, 2021.

[41] Shanshi Huang, Hongwu Jiang, Shimeng Yu, "Mitigating Adversarial Attack for Compute-in-Memory Accelerator Utilizing On-chip Finetune," in *IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2021.

[42] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, Yuan Xie, "PRIME: A Novel Processing-in-memory Architecture for Neural Network," in *International Symposium on Computer Architecture*, Seoul, 2016.

[43] Win-San Khwa, Yen-Cheng Chiu, Chuan-Jia Jhang, Sheng-Po Huang, Chun-Ying Lee, Tai-Hao Wen, Fu-Chun Chang, Shao-Ming Yu, Tung-Yin Lee, Meng-Fan Chang, "A 40-nm, 2M-Cell, 8b-Precision, Hybrid SLC-MLC PCM Computing-in-Memory Macro with 20.5-65.0 TOPS/W for Tiny-Al Edge Devices," in *IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, 2022.

[44] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," in *arXiv*, 2017.

[45] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, Jian Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *Conference on Computer Vision and Pattern Recognition*, Salt Lake City, 2018.

[46] Shuang Wu, Guoqi Li, Feng Chen, Luping Shi, "Training and inference with integers in deep neural networks," in *International Conference on Learning Representations (ICLR)*, Vancouver, 2018.

[47] Jacob, Benoit, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko, "Quantization and training of neural

networks for efficient integer-arithmetic-only inference," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, 2018.

[48] M. Courbariaux, Y. Bengio, J.-P. David, "Training deep neural networks with low precision multiplications," in *workshop contribution at International Conference on Learning Representations (ICLR)*, San Diego, 2015.

[49] Ron Banner, Itay Hubara, Elad Hoffer, Daniel Soudry, "Scalable Methods for 8-bit Training of Neural Networks," in *Neural Information Processing Systems (NIPS)*, Montreal, 2018.

[50] I. Hubara, M. Courbariaux, D Soudry, R. El-Yaniv, Y. Bengio, "Binarized Neural Networks," in *30th Conference on Neural Information Processing Systems (NIPS)*, Barcelona, 2016.

[51] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *European Conference on Computer Vision (ECCV)*, Amsterdam, 2016.

[52] Yukuan Yang, Lei Deng, Shuang Wu, Tianyi Yan, Yuan Xie, Guoqi Li, "Training high-performance and large-scale deep neural networks with," *Neural Networks,* vol. 125, pp. 70-82, 2020.

[53] Jian-Wei Su, Xin Si, Yen-Chi Chou, Ting-Wei Chang, Wei-Hsing Huang, et al., "A 28nm 64Kb Inference-Training Two-Way Transpose Multibit 6T SRAM Compute-in-Memory Macro for AI Edge Chips," in *International Solid- State Circuits Conference*, San Francisco, 2020.

[54] Xin Si, Jia-Jing Chen, Yung-Ning Tu, Wei-Hsing Huang, Jing-Hong Wang, Yen-Cheng Chiu, et al., "A Twin-8T SRAM Computation-In-Memory Macro for Multiple-Bit CNN-Based Machine Learning," in *International Solid- State Circuits Conference*, San Francisco, 2019.

[55] Ming Cheng, Lixue Xia, Zhenhua Zhu, Yi Cai, Yuan Xie, Yu Wang, Huazhong Yang, "TIME: A Training-in-Memory Architecture for RRAM-Based Deep Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 834-847, 2019.

[56] Song Han, Huizi Mao, William J Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *International Conference on Learning Representations*, San Juan, 2016.

[57] Hanbo Sun, Zhenhua Zhu, Yi Ca, Xiaoming Chen, Yu Wang, Huazhong Yang, "An Energy-Efficient Quantized and Regularized Training Framework For Processing-In-Memory Accelerators," in *Asia and* South *Pacific Design Automation Conference (ASP-DAC)*, Beijing, 2020.

[58] Xiaochen Peng, Shanshi Huang, Hongwu Jiang, Anni Lu, Shimeng Yu, "DNN+NeuroSim V2.0: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators for On-Chip Training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2306-2319, 2021.

[59] Wantong Li, Shanshi Huang, Xiaoyu Sun, Hongwu Jiang, Shimeng Yu, "Secure-RRAM: A 40nm 16kb Compute-in-Memory Macro with Reconfigurability, Sparsity Control, and Embedded Security," in *Custom Integrated Circuits Conference*, Austin, 2021.

[60] Anni Lu, Xiaochen Peng, Wantong Li, Hongwu Jiang, Shimeng Yu, "NeuroSim Simulator for Compute-in-Memory Hardware Accelerator: Validation and Benchmark," *frontiers in Artificial Intelligence*, 2021.

[61] Shihui Yin, Yulhwa Kim, Xu Han, Hugh Barnaby, Shimeng Yu, Yandong Luo, Wangxin He, Xiaoyu Sun, Jae-Joon Kim, Jae-sun Seo, "Monolithically Integrated RRAM- and CMOS-Based In-Memory Computing Optimizations for Efficient Deep Learning," *IEEE Micro*, vol. 39, no. 6, pp. 54 - 63, 2019.

[62] Yun Long, Xueyuan She, Saibal Mukhopadhyay, "Design of Reliable DNN Accelerator with Un-reliable ReRAM," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, 2019.

[63] Hongwu Jiang, Xiaochen Peng, Shanshi Huang, Shimeng Yu, "CIMAT: A Compute-In-Memory Architecture for On-chip Training Based on Transpose SRAM Arrays," *IEEE Transactions on Computers* , vol. 69, no. 7, pp. 944 - 954, 2020.

[64] Jiyong Woo, Shimeng Yu, "Resistive Memory-Based Analog Synapse: The Pursuit for Linear and Symmetric Weight Update," *IEEE Nanotechnology Magazine* , vol. 12, no. 3, pp. 36-44, 2018.

[65] Matthew Jerry, Pai-Yu Chen, Jianchi Zhang, Pankaj Sharma, Kai Ni, Shimeng Yu, Suman Datta, "Ferroelectric FET analog synapse for acceleration of deep neural network training," in *IEEE International Electron Devices Meeting (IEDM)*, San Francisco, 2017.

[66] Shinhyun Choi, Scott H. Tan, Zefan Li, Yunjo Kim, Chanyeol Choi, Pai-Yu Chen, Hanwool Yeon, Shimeng Yu, Jeehwan Kim, "SiGe epitaxial memory for neuromorphic computing with reproducible high performance based on engineered dislocations," *Nature Materials,* vol. 17, pp. 335-340, 2018.

[67] Wei Wu, Huaqiang Wu, Bin Gao, Peng Yao, Xiang Zhang, Xiaochen Peng, Shimeng Yu, He Qian, "A Methodology to Improve Linearity of Analog RRAM for Neuromorphic Computing," in *IEEE Symposium on VLSI Technology*, Honolulu, 2018.

[68] Jianshi Tang, Douglas Bishop, Seyoung Kim, Matt Copel, Tayfun Gokmen, Teodor Todorov, SangHoon Shin, Ko-Tao Lee, Paul Solomon, Kevin Chan, Wilfried Haensch, John Rozen, "ECRAM as Scalable Synaptic Cell for High-Speed, Low-Power Neuromorphic Computing," in *IEEE International Electron Devices Meeting (*IEDM*)*, San Francisco, 2019.

[69] Roel MAES, Physically Unclonable Functions, Springer-Verlag Berlin Heidelberg, 2013.

[70] Xiaochen Peng, Rui Liu, Shimeng Yu, "Optimizing Weight Mapping and Data Flow for Convolutional Neural Networks on RRAM Based Processing-In-Memory Architecture," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Sapporo, 2019.

[71] Ian Goodfellow,Jonathon Shlens,Christian Szegedy, "Explaining and Harnessing Adversarial Examples," in *International Conference on Learning Representations* , San Diego, 2015.

[72] Joan Bruna, Christian Szegedy, Ilya Sutskever, Ian Goodfellow, Wojciech Zaremba, Rob Fergus, Dumitru Erhan, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, Banff\, 2014.

[73] Nicholas Carlini, D. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *IEEE Symposium on Security and Privacy (SP)*, San Jose, 2017.