Bias-mapped Computation-In-Memory Neural Inference Engine using RRAMs

By

VARUN SUDHAKAR
4547020

DEPARTMENT OF COMPUTER ENGINEERING
FACULTY OF ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE
TECHNISCHE UNIVERSITEIT DELFT

A thesis submitted to the Delft University of Technology in accordance with the requirements of the degree of MASTER OF SCIENCE in Computer Engineering to be publicly defended on 13th May 2022.

13th May 2022

Abstract

The ever-increasing energy demands of traditional computing platforms (CPU, GPU) for large-scale deployment of Artificial Intelligence (AI) has spawned an exploration for better alternatives to existing von-Neumann compute architectures. Computation In-Memory (CIM) using emerging memory technologies such as Resistive Random Access Memory (RRAM) provide an energy-efficient and scalable alternative for Deep Neural Networks (DNN) applications. However, the benefits of CIM frameworks come at the cost of low DNN accuracy due to non-idealities in RRAM devices. In this thesis we address the conductance variation non-ideality in RRAM devices at an architectural level. We present two mapping schemes to improve the accuracy of CIM-based DNNs in the presence of RRAM conductance variation. Experimentation conducted with five datasets show that all proposed schemes provide up to 5.4x accuracy improvement over state-of-the-art implementations, while inducing a 1.5% area cost and up to 10% energy overhead. Based on accuracy-energy trade-off, the thesis concludes the proposed Complementary Conductance Matrix (CCM) is the best candidate to improve inference accuracy of neural networks on CIM hardware using RRAM. It reports an accuracy improvement up to 5x with 1.52% area overhead and 9% energy overhead.

Acknowledgements

I would like to thank Prof. Rajendra Bishnoi for his constant encouragement, support and guidance throughout the research process. He steered my thesis in the right direction with clarity and detailed inputs to help improve my research and build my knowledge in the field of in-memory computing. With his deep expertise in this field, Prof. Said Hamdioui provided invaluable advice in improving my thesis work, for which I would like to express my gratitude. I would like to thank Sumit Diware for his advice, explaining concepts, resolving doubts and help on innumerable occasions. I would also like to acknowledge the computer engineering group for their feedback during my thesis work.

I am eternally grateful to my parents for their unconditional love, support and guidance while giving me an opportunity to grow. I am fortunate to have shared incredible memories with my friends and look forward to many more. Without these people, none of this would have been possible.

Delft, May 2022
Varun Sudhakar

Table of Contents

Acronyms

ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
BL	BitLine
C2C	Cycle-to-Cycle
CCM	Complementary Conductance Matrix
CF	Conductive Filament
CIM	Computation In-Memory
CMOS	Complementary Metal-Oxide Semiconductor
CNN	Convolutional Neural Network
D2D	Device-to-Device
DAC	Digital-to-Analog Converter
DNN	Deep Neural Networks
DRAM	Dynamic Random Access Memory
DSC	Dynamic Scaling Column
DSF	Dynamic Scaling Factor
FCN	Fully Connected Networks
FP	Floating-Point
FXP	Fixed-point
HCS	High Conductance State
ICS	Intermediate Conductance States
LCS	Low Conductance State
LSTM	Long Short-Term Memory

MAC	Multiply-Accumulate
MLC	Multi-Level Cell/Operations
MLP	Multi-Layer Perceptron
NVM	Non-Volatile Memory
NVMs	Non-Volatile Memories
PCM	Phase Change Memory
RRAM	Resistive Random Access Memory
RTN	Random Telegraph Noise
SAF	Stuck-At Faults
SRAM	Static Random Access Memory
STT-RAM	Spin-Transfer Torque Magnetic Random Access Memory
VMM	Vector Matrix Multiplication
VMs	Volatile-Memories
WL	WordLine

List of Figures

List of Tables

CHAPTER 1

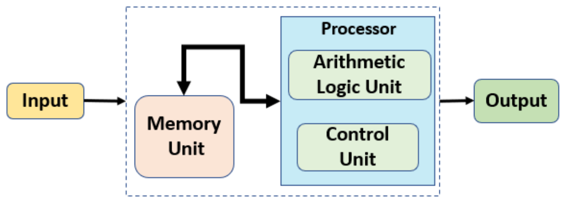
Introduction

This chapter introduces the thesis. Context and motivation for this research is first presented from a computing and Artificial Intelligence (AI) perspective. This is followed by a detailed discussion of the problem statement. Next, the thesis contributions are presented after which the chapter concludes by outlining the organization of this thesis.

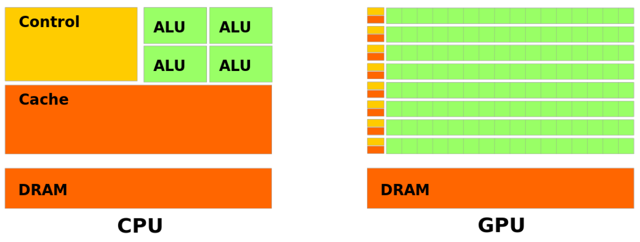
Conventional von-Neumann architectures suffer from the Memory Wall. Additionally, limitations on Complementary Metal-Oxide Semiconductor (CMOS) transistor scaling and high power leakage of CMOS charge-based memories and circuits has seen the emergence of Computation In-Memory (CIM) architectures. CIM with Non-Volatile Memory (NVM) technologies such as Resistive Random Access Memory (RRAM) present a scalable alternative for Artificial Intelligence (AI) applications.

1.1 Computing and AI

The emergence of Artificial Intelligence (AI) and its reliance on large datasets as a critical component across various sectors *viz.* genomics, robotics, medical predictions, self-driving, financial analytics and recognition systems has exposed the limitations of von-Neumann architectures.



(a) Traditional von-Nuemann Architecture



(b) CPU vs GPU architecture

Figure 1.1: CPU and GPU based on traditional von-Nuemann architectures rely on data communication between processing and memory units [1]

- Memory Wall [14]: The diverging performance gap between the processor and memory unit has resulted in idle processor cycles waiting for the memory to provide data. This limited data bandwidth has also been termed as the von-Neumann bottleneck.

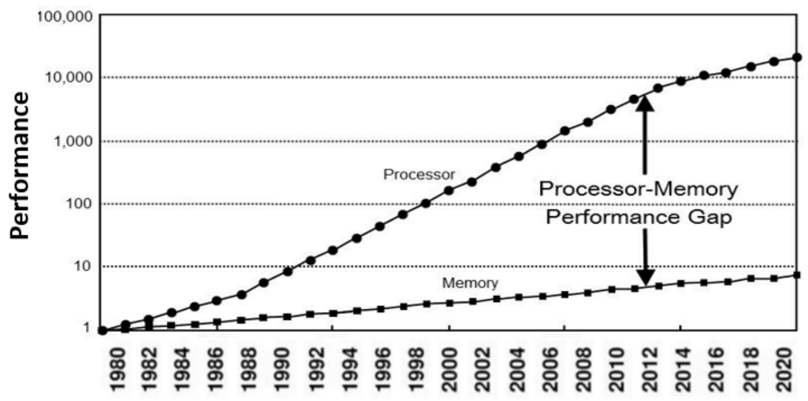
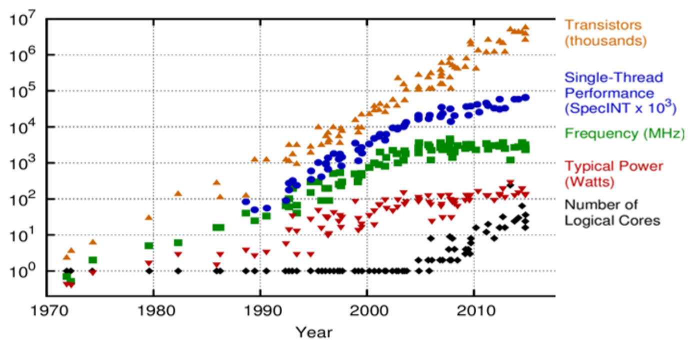


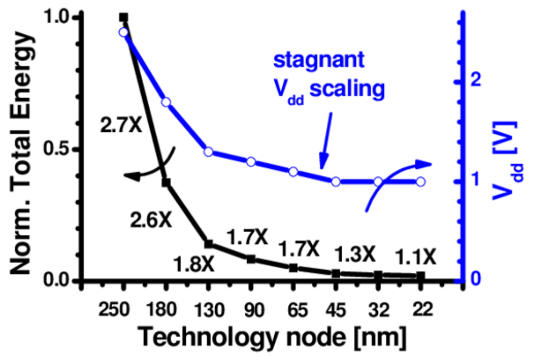
Figure 1.2: Increasing performance gap between memory and processor over the decades has resulted in a Memory wall, also known as the von-Neumann bottleneck.

- Power Wall: The increasing number of CMOS transistors on electronic chips (Moore's Law) is not sustainable with current compute architectures since we have already reached our power and cooling budget. Dynamic power ($P_{dynamic} = A \cdot C \cdot V_{dd}^2 \cdot f$) has increased due to stagnant V_{dd} (supply voltage) scaling and increased clock frequency (f). Static power leakage ($P_{static} = I_{leakage} \cdot V_{dd}$) due to current leakage from sub-threshold and gate oxide leakage is also a concern.

- CMOS scaling: CMOS scaling below 10nm has resulted in high static power loss, leakage current, low yield and reliability [15]. Furthermore, this will not economically viable due to the increasing cost of fabrication and testing



(a) More transistors on chip along with increasing operational frequency has increased the required power budget. [16]



(b) Dynamic power increases as V_{dd} has stagnated with smaller technology nodes. [17]

Figure 1.3: Traditional CMOS computing architectures have already surpassed their power and cooling budget resulting in the Power wall.

Unconventional memory-centric compute architectures look to overcome the Memory Wall, CMOS scaling limitations using Non-Volatile Memory (NVM). Computation In-Memory (CIM) has emerged as a promising candidate for large-scale, efficient AI workloads [18–20].

1.2 Computation In-Memory (CIM) for AI workloads

CIM overcomes the von-Neumann bottleneck (data transfer between processor and memory) by integrating Non-Volatile Memories (NVMs) that enable data storage and computation both within the memory itself. CIM architectures replace digital operations such as multiplication and addition with analog computation using the principles of Ohm's law and Kirchoff's current law. In-Situ computation using NVMs allows CIM architectures to achieve higher energy-efficiency compared to traditional computing platforms based on von-Neumann architectures [18–22].

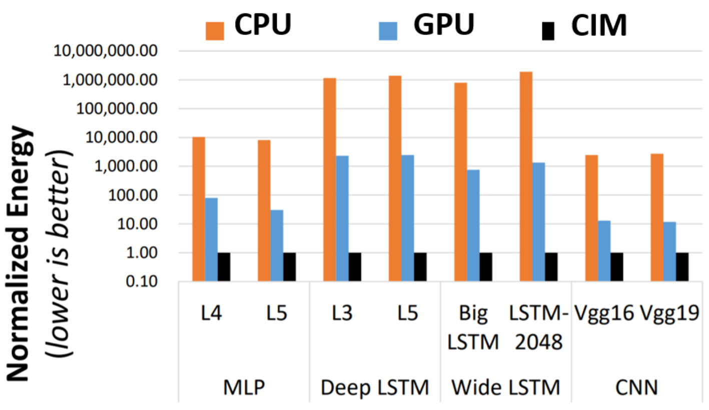
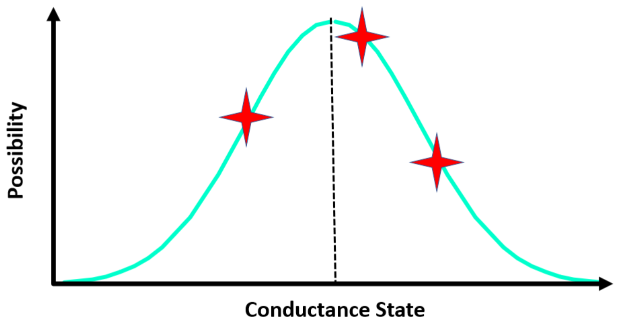
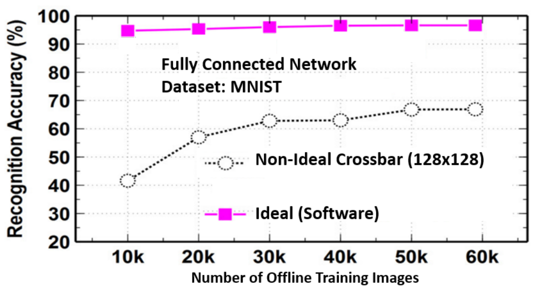


Figure 1.4: Demonstration of energy-efficient CIM architectures across different networks such as Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) [2]

Significant research interest in hardware for AI has seen a shift in focus from conventional CMOS charge-based memories *viz.* Static Random Access Memory (SRAM), Dynamic Random Access Memory (DRAM) and Flash, also known as Volatile-Memories (VMs). Scaling these CMOS-VMs below 10nm has resulted in high static power leakage, low wafer yield and reliability issues [15, 23, 24]. Hence, emerging memory technologies such as Non-Volatile Memories (NVMs) are seen as a suitable alternative for modern AI tasks [25, 26]. NVMs, also known as memristors, such as Spin-Transfer Torque Magnetic Random Access Memory (STT-RAM) [27–29], Phase Change Memory (PCM) [12, 30] and Resistive Random Access Memory (RRAM) [6, 24, 31–37] provide significant advantages such as zero static power leakage, high cell density and long retention. Furthermore, NVMs can be easily integrated into current CMOS manufacturing processes and only needs a minimal redesign of the memory cell [20]. However, STT-RAM and PCM face significant scalability and design challenges, hence RRAM-based CIM architectures are seen as a better choice for large-scale crossbar implementations for AI tasks.



(a) Devices on the crossbar hold different conductance states under ideal conditions [38]



(b) Degradation of accuracy due to non-ideal RRAMs due to variation [39]

Figure 1.5: CIM-based architectures obtain reduced accuracy on classification tasks since NVMs in the same crossbar array and programmed under identical conditions, achieve different conductance states.

RRAM-based CIM architectures suffer from accuracy degradation due to various non-idealities in RRAM devices. One of the prominent RRAM non-idealities is conductance variation. It refers to the variation of conductance states across different RRAMs under identical programming conditions across a crossbar array. In this thesis, we implement a bias-mapped CIM Neural engine using RRAMs that achieves high accuracy by minimizing the impact of variability using conductance mapping techniques.

1.3 Problem Statement

- Traditional von-Neumann compute architectures (CPUs/GPUs) have been integral to the development and large-scale adoption of AI across a wide range of applications. However, the memory wall and CMOS scaling and leakage limitations require us to move beyond the von-Neumann computing paradigm. Memory-centric compute architectures that reduce data transfer are a promising alternative.
- Resistive Random Access Memory (RRAM)-based Computation In-Memory (CIM) hardware architectures provide an energy-efficient, low-latency parallel computing paradigm that can be easily integrated into CMOS fabrication and manufacturing processes. However, CIM-based neural networks suffer from degradation of accuracy owing to RRAM non-idealities such as conductance variation.
- Innovative non-ideality mitigation techniques are required to ensure high neural network accuracy on CIM architectures for AI applications.

1.4 Contributions

This thesis presents novel techniques for mapping the trained neural network weights to RRAM-based CIM architectures in order to improve the classification accuracy of neural networks in the presence of conductance variations. The key contributions of this thesis are:

- **Two Crossbar Conductance Mapping Schemes**

  We implement two conductance mapping schemes that leverage the asymmetry conductance variation profile of RRAM-based CIM frameworks.

  1. ***Dynamic Scaling Column (DSC)***

     An extra column per weight is introduced into the crossbar array to dynamically reduce the error during reconciliation of the partial outputs and consequently minimizes the error in matrix operations.

  2. ***Complementary Conductance Matrix (CCM)***

     To ensure low variability of RRAMs mapped onto a crossbar array, we use a hardware-software co-optimization to extract a G_{max} complement of the original conductance matrix. This complementary matrix is mapped onto the crossbar array through a differential weight representation to preserve and improve accuracy.

- **Performance Evaluation of Mapping Schemes**

  A comprehensive evaluation of CCM and DSC compared to state-of-the-art mapping techniques is performed. Across various datasets, we measure accuracy, energy and area statistics to delineate the improvements achieved by the proposed schemes. Furthermore, the analysis of the combined methodology (CCM + DSC) is compared to state-of-the-art implementations.

Proposed Dynamic Scaling Column (DSC) presents an accuracy improvement up to 2.85x by incurring a 1.53% area and 9.8% energy overhead whereas the proposed Complementary Conductance Matrix (CCM) achieves an accuracy improvement up to 5x incurring 1.52% area and 9% energy overhead. Based on trade-offs between accuracy and energy, CCM is the best suited mapping technique for RRAM-based Neural Inference Engine on bias-mapped CIM hardware.

1.5 Organization

The rest of the thesis is organized as follows:

- **Chapter 2** provides a detailed background of concepts relevant to this thesis. We discuss fundamentals of Neural Networks, CIM frameworks and NVMs. Further, we compare different NVM technologies to understand the motivation behind investigating RRAM-based CIM architectures

- **Chapter 3** discusses state-of-the-art works that implement mapping schemes to improve accuracy. Analysis of key contributions and drawbacks of these works are presented
- **Chapter 4** begins by outlining the implementation to realize mapping onto a CIM crossbar array. It further explains the salient aspects of the proposed mapping schemes, *i.e.* Dynamic Scaling Column (DSC) and Complementary Conductance Matrix (CCM).
- **Chapter 5** presents the simulation setup, followed by discussion of the results and analysis based on design metrics (accuracy, energy, area) for the RRAM-based Neural Engine CIM classifier.
- **Chapter 6** concludes the thesis by highlighting the trade-offs and making suggestions for future work.

CHAPTER 2

Background

This chapter provides the background information required for this thesis. In Section 2.1, the fundamental architecture and working of Deep Neural Networks (DNN) is explained. Further, Section 2.2 outlines the Computation In-Memory (CIM) architecture. Section 2.3, an in-depth analysis of Resistive Random Access Memory (RRAM) devices, concluding with a summary in Section 2.4

2.1 Deep Neural Networks

This section provides a fundamental understanding of Deep Neural Networks (DNN). DNN architectures *viz.* Fully Connected Networks (FCN), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) consists of neurons that process digital information (audio, video, text) to realize various large-scale real world implementations in AI tasks such as genomics, recognition systems, self-driving, finance and robotics. However, for this thesis we will consider Fully Connected Networks (FCN) architectures. Internal workings of Fully Connected Networks (FCN) from a mathematical perspective are discussed and followed by concepts of training and inference (with a CIM perspective).

2.1.1 Fully Connected Networks (FCN)

The Fully Connected Networks (FCN) learns to understand data by feeding it through multiple layers of neurons. A schematic of a Fully Connected Networks (FCN) is presented in Figure 2.1. In a FCN, all neurons have edges (weights) to all the activation outputs of the previous layer.

Input Layer —This layer receives the digital input. Data normalization and standardization is performed before feeding the data (Matrix $A = [A_1, A_2, A_3, ..., A_M]$) to the subsequent layer (Layer $L+1$).

Hidden Layer —Neurons in the hidden layer receive the data along with the *weight* (Matrix $W = [W_1, W_2, W_3, ..., W_M]$) of the data. Weights signify the importance of the data being processed in the neuron. In the neuron, input data is multiplied with the weight and a external *bias* is added to the result. The weighted sum of resulting values are processed by an activation function f. These values are fed into the next layer. The primary operation of multiplying inputs and weights is called Vector Matrix Multiplication (VMM)

$$B = f(W_{A\to B}^{T} A + bias) \tag{2.1}$$

Output Classification Layer —The number of neurons in this layer depends on the number of output classes defined/required for a specific DNN task. The output is computed similar to the hidden unit: $C = f(W_{B\to C}^{T} B + bias)$. Various activation functions, such as sigmoid, ReLU, tanh can be used for each layer. Finally, a softmax used to ensure all outputs are normalized between (0, 1) to extract the probability distribution of the prediction of the DNN.

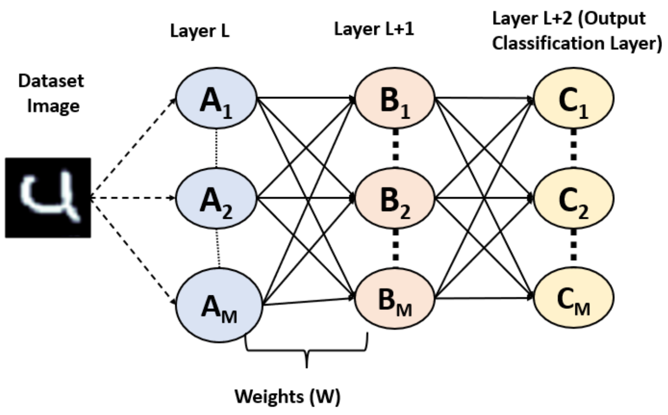


Figure 2.1: Example of a Fully Connected Networks (FCN) where the dataset is fed into the input layer, processed by the hidden layer of neurons and produces a classification distribution at the output layer.

2.1.2 Training and Inference

Deep Neural Networks (DNN) operate in training and inference (validation, evaluation) mode. Training of DNNs is an iterative optimization problem where tunable parameters (neuron weight, learning rate) are used to assist neural networks to learn from data. Training of DNNs can be performed in a supervised or unsupervised manner. In supervised training, the neural network requires the expected number of output classes for a given input. On the other hand, unsupervised training expects the neural network to understand the data without any external stimulus to assist the network in learning. A pitfall of DNN training is to ensure the network extracts the feature representation (latent vector space) of the data without simply memorizing the dataset. This is known as overfitting and can be effectively mitigated through data augmentation techniques (rotate, crop, blur). This thesis adopts a supervised training method with data augmentation techniques.

Training involves three components:

Forward Pass → Data flows through the neurons till the output layer where the prediction error is computed compared to the ground truth (Boolean value indicating the ideal output).

Backward Propagation → This error is propagated backwards (backprop) through the DNN to compute the gradients of weights and other parameters. This enables the DNN to adjust the tunable parameters of the neuron to improve the next prediction.

Weight Update → Based on gradient values, learning rate, regularizer and other parameters, the weight of the neuron is updated to improve the overall accuracy of the DNN

During training, when the DNN completes a single cycle of these tasks, this is an *epoch*. DNNs iteratively predict data through each epoch till it achieves a high classification accuracy for a task. Next step is inference.

Inference on the DNN is performed on unseen data after the neural network is optimized over the training data. The neurons are preset with the optimized weights extracted during training and the DNN performs predictions on "in-the-wild" data. DNN prediction only requires a forward pass to classify the data. Inference is less time and energy consuming compared to training deep neural networks.

2.1.2.1 In-Situ (Online) and Ex-Situ (Offline) Training in CIM architectures

The iterative weight optimization method (Forward pass, backprop, weight update) of Deep Neural Networks (DNN) implemented onto the crossbar array by mapping and updating weights is known as In-Situ (Online training), *i.e.*, the entire DNN training process runs on-chip. However, CIM architectures are significantly limited in training DNNs. This is due to their inefficiency in Floating-Point (FP) arithmetic computation, which is a critical aspect during Back-Propagation to compute gradients to indicate adjustments required in the neuron. As a consequence, inaccurate convergence for weights (and weight updates) in the DNN reduce accuracy.

For Ex-Situ (offline training), the training is done off-chip on traditional von-Neumann architecture to extract the optimized weights. These optimized weights of the DNN are mapped onto the crossbar array to perform inference. CIM is fundamentally suited for Fixed-point (FXP) arithmetic required in forward pass. Since no weight updates are required during inference, CIM architectures present a scalable and efficient computing platform for DNN inference engines.

2.2 Computation In-Memory

This section begins by outlining the fundamental advantage of Computation In-Memory (CIM) architectures for AI workloads. Next, its inner workings from a computation perspective are presented. Finally, training and inference is discussed within the context of CIM architectures.

2.2.1 Architecture

Computation In-Memory (CIM) is a promising compute architecture to overcome the Memory wall faced by existing von-Neumann architectures and CMOS scaling and leakage issues . CIM integrates processing within the memory core itself. This reduces the transfer of data to perform computations. Computation is performed within the memory core. However, to realize logic and arithmetic operations, extra circuitry is required. The computation current output is produced in the peripheral circuit [40].

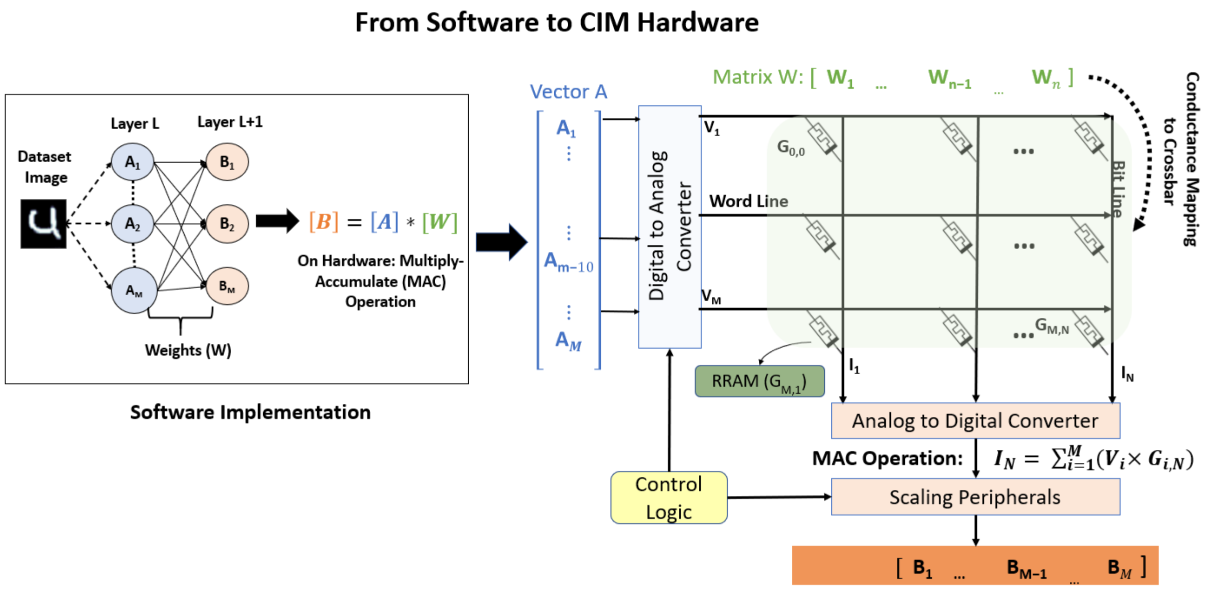

From Software to CIM Hardware
Layer L
Layer L+1
Dataset Image
A_1
A_2
A_M
B_1
B_2
B_M
Weights (W)
$[B] = [A] * [W]$
On Hardware: Multiply-Accumulate (MAC) Operation
Software Implementation
Vector A
A_1
A_{m-10}
A_M
Digital to Analog Converter
Matrix W: [W_1 ... W_{n-1} ... W_n]
V_1
$G_{0,0}$
Word Line
Bit Line
V_M
$G_{M,N}$
RRAM ($G_{M,1}$)
I_1
I_N
Conductance Mapping to Crossbar
Analog to Digital Converter
MAC Operation: $I_N = \sum_{i=1}^{M}(V_i \times G_{i,N})$
Scaling Peripherals
Control Logic
[B_1 ... B_{M-1} ... B_M]

Figure 2.2: Visualization of the mapping from a software implementation of the neural network to a CIM based framework.

Vector Matrix Multiplication (VMM) for DNNs is realized on hardware through Multiply-Accumulate (MAC) operations. The programmed conductance represent weights of the DNN. At the intersection of each WordLine (WL) and BitLine (BL), RRAMs through Ohm's Law, produce the product of an input voltage and the conductance.

$$I = V \cdot G$$

At the end of each BitLine (BL), the additional of currents through Kirchoff's current law produce the partial current outputs. These currents are converted to digital data through ADCs, shifted and summed to produce the result of the Vector Matrix Multiplication (VMM).

$$I_N = \sum_{i=1}^{M} (V_i \times G_{i,N})$$

Using analog periphery such as Digital-to-Analog Converter (DAC) and Analog-to-Digital Converter (ADC), CIM architectures can implement Vector Matrix Multiplication (VMM) [41]. Multiplication and addition operations are performed through Ohm's law and Kirchhoff's law respectively. Additionally, CIM circuit designs require minimal resign of the memory array since the design optimization focus is on CMOS based peripheral circuits [20].

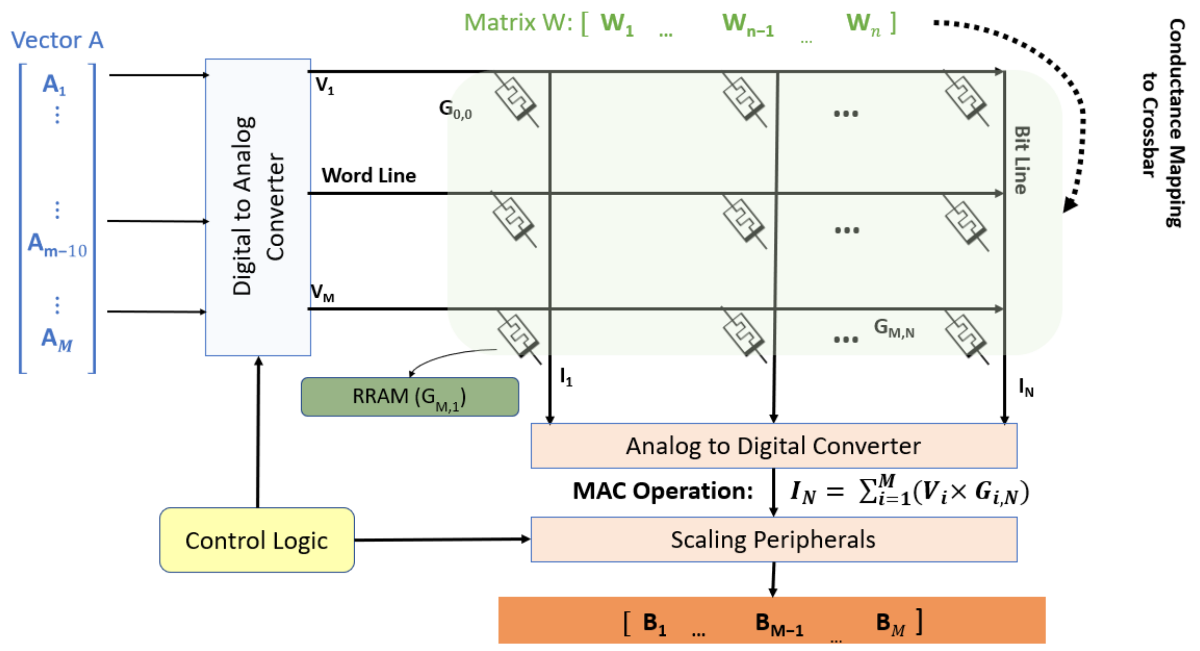

Vector A
A_1
A_{m-10}
A_M
Matrix W: [W_1 ... W_{n-1} ... W_n]
Conductance Mapping to Crossbar
Digital to Analog Converter
V_1
$G_{0,0}$
Word Line
V_M
Bit Line
$G_{M,N}$
RRAM ($G_{M,1}$)
I_1
I_N
Analog to Digital Converter
MAC Operation: $I_N = \sum_{i=1}^{M}(V_i \times G_{i,N})$
Scaling Peripherals
Control Logic
[B_1 ... B_{M-1} ... B_M]

Figure 2.3: Schematic of the complete CIM architecture for VMM computation. Using Ohm's Law and Kirchoff's Law, MAC operations are performed in the crossbar array and the result is produced in the peripheral circuits

2.3 Non-Volatile Memories for CIM

As a consequence of the memory wall and CMOS scaling limitations, the research community has looked towards revising requirements on cell density and energy consumption. This will significantly reduce the cost of manufacturing and fabricating high performance computing machines. Hence, existing memory technologies such as Static Random Access Memory

(SRAM), Dynamic Random Access Memory (DRAM) and Flash are getting out of favour for data-centric applications. Conventional CMOS memories suffer from increased power leakage and lithography scalability issues below 10nm. SRAMs provide fast random access but have low density. DRAMs have higher density but slower than SRAMs. Flash memories suffer from very high write cost to negate the high density compared to DRAMs.

This has seen a shift to Non-Volatile Memories (NVMs) that provide high cell density, low power consumption and quick operation comparable to current processors. NVMs are also referred to as memristors. Various memristors such as Resistive Random Access Memory (RRAM), Spin-Transfer Torque Magnetic Random Access Memory (STT-RAM), Phase Change Memory (PCM) etc. already exist [26, 42]. RRAMs are a scalable solution for Deep Neural Networks (DNN) applications do not rely on thermal (PCM) or magnetic (STT-RAM) programming of the memristor. This thesis deals with RRAMs.

2.3.1 RRAM Principles and Operation

RRAMs enable CIM architectures to perform DNN computation. It stores memory through a non-linear conductive element. Leon Chua [43], in 1971, first hypothesized that there was a missing non-linear circuit element to characterize the relationship between charge ($q(t)$) and magnetic-flux-linkage ($\phi(t)$): $M = \frac{d\phi(t)}{dq(t)}$ which can also be represented as:

$$\frac{\frac{d\phi(t)}{dt}}{\frac{dq(t)}{dt}} = \frac{V(t)}{I(t)}$$

In 2008, Hewlett-Packard Labs demonstrated the memristor as a fundamental circuit element.

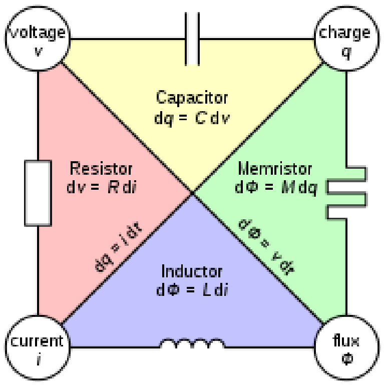


Figure 2.4: The fourth missing fundamental circuit element —Memristor. It defines the relationship between charge and magnetic flux linkage [3]

RRAMs have a distinct fingerprint that can be associated with their *I-V* hysteresis behavior [44]. Figure 2.5 depicts the pinched *I-V* hysteresis loop characteristic to memristors. This is also known as *bipolar switching* behaviour. This indicates that the switching direction [45]. This implies that the switching behaviour depends on the polarity of the voltage applied to the RRAM device.

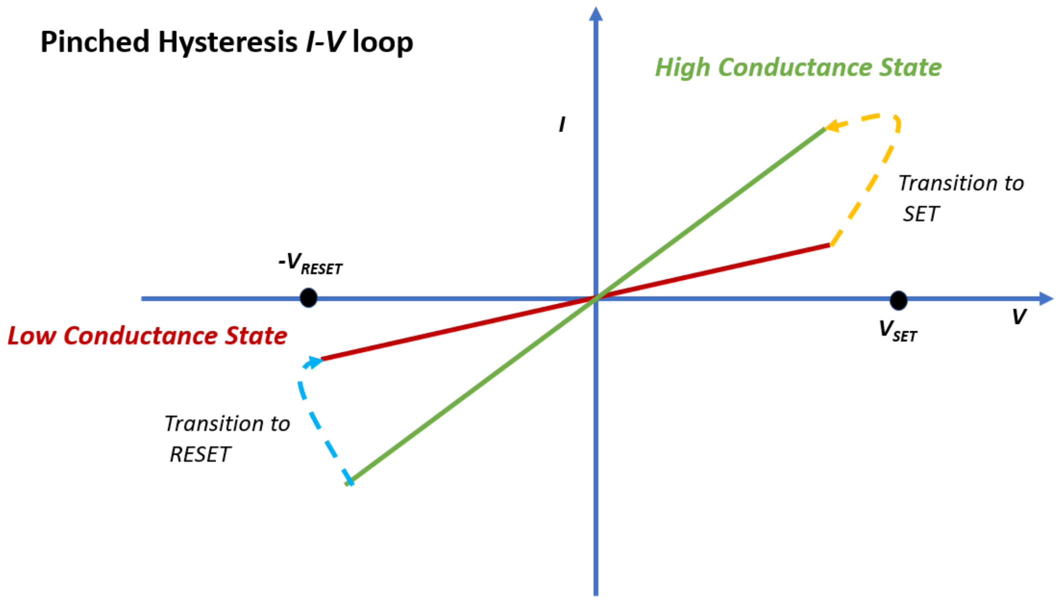


Figure 2.5: Fingerprint of RRAM Memristors: A pinched *I*-*V* hysteresis loop depicting bipolar switching behaviour

$-V_{RESET}$ indicates the polarity and voltage needed to switch a memristor from High Conductance State (HCS) to Low Conductance State (LCS). This is also known as the RESET transition. V_{SET} indicates the voltage required to switch a memristor from a Low Conductance State (LCS) to a High Conductance State (HCS). This process is known as SET transition.

The stability of switching characteristics between conductance states depends on the device physics and its stochastic behaviour. Ideally, memristors should store analog values at a steady state but practically exhibit variable intermediate states (Multi-Level Cell/Operations (MLC)). Programming the conductance states of a memristor is influenced by the magnitude and duration (pulse) of the applied voltage. However, it is important to limit the current (Compliance current: I_{cc}) or apply small voltages since the switching endurance of the device degrades with applied voltage stimulus.

Resistive Random Access Memory (RRAM)s work and store data by modulating the conductance across a solid-state dielectric (switching medium) material [6, 24, 31–37]. The switching mechanism from LCS to HCS is the SET process and switching from HCS to LCS is the RESET process. The switching mechanism is based on the formation/breakdown of the conductive filament when a voltage is applied. However, to avoid permanent dielectric breakdown resulting in Stuck-At Faults (SAF), a SET compliance current (I_{cc}) is enforced.

In the SET process, dielectric breakdown occurs triggering oxygen ios to drift to the bottom electrode through the electric field and discharged as a neutral non-lattice oxygen ion. This results in an oxygen reservoir at the bottom electrode-oxide interface. When the RRAM is in the HCS current flows in the bulk oxide through the Conductive Filament (CF). In the RESET process, oxygen ions migrate to the bulk oxide to recombine with the oxygen vacancies to return the RRAM to its original LCS state [4, 46].

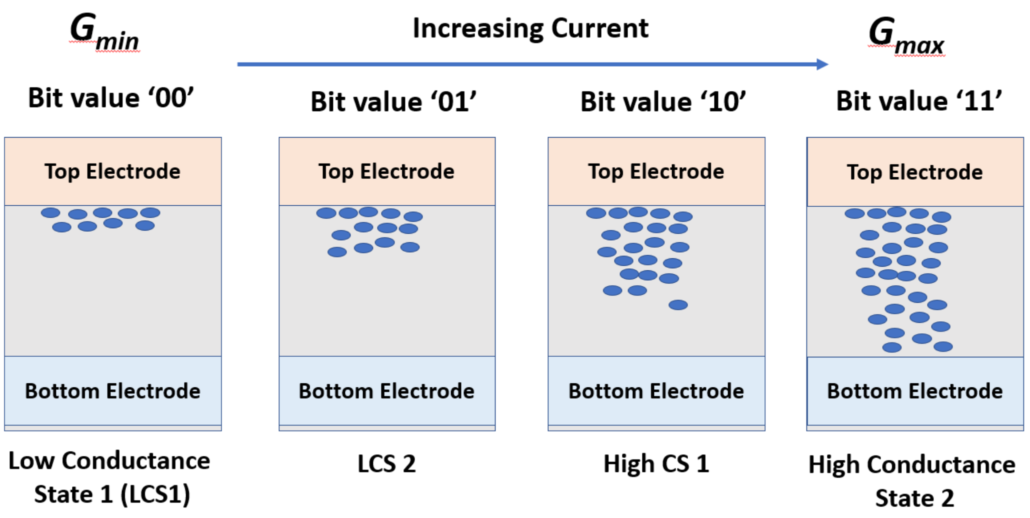


Figure 2.6: Multi-Level Cell/Operations (MLC) exhibited by RRAMs. By successively applying voltage pulses, we can programs specific conductance values to the device [4, 5]

The top electrode controls the diffusion of oxygen vacancies to form a conductive filament. The conductivity is determined by the distance to the bottom electrode and distribution of oxygen vacancies within the switching material. To read values from an RRAM, small bias voltage is applied to the device to read out currents. Writing values to the RRAM device involves a process called Closed-Loop programming [47–49]. This is an iterative process to ensure the RRAM holds a steady conductance state of the desired programmed value. Hence in order to program a value to an RRAM, successive voltage pulses are applied to the device and read out to verify the state of the device. This method is also known as Program-Read-Verify.

However, this write mechanism enables programming to Intermediate Conductance States (ICS). This is known as Multi-Level Cell/Operations (MLC) [33, 46]. This enables RRAMs to store more than 1 bit of data. Figure 2.6 shows a 2 bit RRAM. By applying successive voltage pulses, oxygen vacancy diffusion increases thereby increasing the current through the Conductive Filament (CF). Researchers have demonstrated RRAMs up to 8 bit resolution.

RRAMs enable high cell density ($4F^2$), low operating voltage, zero power leakage and long data retention and high switching endurance . Furthermore, RRAMs do not rely on thermal (PCM) or magnetic (STT-RAM) programming of the memristor and hence is a scalable solution for Deep Neural Networks (DNN) applications. This thesis adopts RRAM devices for CIM.

However on a device and circuit level, RRAM-based CIM architectures present some challenges:

- **Device Level Non-Idealities**
  - Conductance Drift: When RRAM store data over long periods of time without being refreshed. A migration of oxygen ions over time occurs leading to decay of the Conductive Filament (CF). This can be mitigated through closed-loop-programming methods to maintain a desired steady state conductance value.
  - High write cost: Since, closed-loop-programming (Program-Read-Verify) is an iterative process, it induces a high cost when writing values to the RRAM. Furthermore, frequent writes to the RRAM reduces the switching endurance of the device and is more prone to Stuck-At Faults (SAF)
  - Read-Disturb: When applying a small bias voltage to the RRAM to read out current values, the voltage application perturbs the value held in the RRAM. This reduces the reliability of the current output in the ADC units due to a reduced sensing margin [50]
- **Circuit Level Non-Idealities**
  - Sneak paths: They are alternative, undesirable current paths that interfere with the required current measurement path. This results in incorrect reading of current outputs and synaptic states [51–54]. Furthermore, magnitude of the sneak currents depend on memristive conductance state. Methods such as read-biasing reduce the severity of sneak paths. but primarily focused towards detection schemes.
  - Wire Non-idealities: They originate from resistances based on thickness and length of the wire which cause a dynamic IR drop and also induces higher electronmigration. Furthermore, the length of the interconnect wires and size of the crossbar array affect the magnitude of the parasitic influence of wire non idealities.
  - Noise is a consequence of thermal properties of the RRAM when actively performing computations [18, 20, 55]. A higher operating temperature (and ambient temperature) increases the rate of oxygen vacancy diffusion. Implementing conductance margins (offset) can cancel out RTN and ensure reliable operation.

Although various non-idealities and mitigation schemes on a device and circuit level have been previously explored, a prominent non-ideality that significantly degrades neural network accuracy is conductance variation.

2.3.2 Conductance Variation

This variability is when multiple RRAM devices hold different conductance states under identical programming conditions on the same crossbar array. These variabilities reduce neural network accuracy for classification tasks. Conductance variabilities are caused due to:

- **Cycle-to-Cycle (C2C) Variations** —This is due to the stochastic (non-deterministic) nature of the formation and breakdown of oxygen vacancies in the Conductive Filament (CF). Furthermore, C2C variability is inversely proportional to the SET compliance current (I_{cc}). However, increasing SET compliance current (I_{cc}) results in a small oxygen vacancy distribution at High Conductance State (HCS), since higher number of defects in the Conductive Filament (CF) forms a well-defined conductive pathway. In other words, due to random locations and defects in the conducting channel, defect migration triggers at different activation energies.

- **Device-to-Device (D2D) Variations** —This variability originates from non-uniformities in the fabrication and baking process resulting in varying thickness of switching material, defects and surface roughness of metal electrodes. Inadequate control over defect generation and Conductive Filament (CF) formation during SET processes, also known as electroforming, enhances thee variabilities. Electroformation induced variations originate from current overshoot due to parasitic effects in the switching material. This current overshoot can vary between devices and lead to undesirable defect generation. Furthermore, when electrons are conducted through the metal, it interacts with Schottky defects and fault-lines. This results in scattering. Since thermal energy produced scattering since it imparts energy causing atoms to vibrate. Hence, higher working temperatures also result in more scattering. Process variations also lead to varying conductance range, retention and switching endurance. These aspects can reduce yield as well as lead to Stuck-At Faults (SAF).

Researchers model the variations of RRAMs following a Gaussian distribution. G_{Final} is the final programmed conductance state into the RRAM. G_{Ideal} is the desired conductance state for a specific application/task. ΔG is a random variable that follows a Gaussian normal distribution.

$$G_{Final} = G_{Ideal} + \Delta G;\ \text{where } \Delta G \approx N(\mu, \sigma)$$

Prakash *et. al.* [6] characterized the variability for a fabricated 2 bit RRAM device. Figure 2.7 depicts the conductance variations for each Intermediate Conductance States (ICS). An asymmetry in the variation distribution is observed. Low Conductance States exhibit a higher conductance variation (σ/μ) compared to High Conductance States. The lowest conductance state has a conductance variation of 20% whereas the highest conductance state exhibits a variation of 2.4%. This asymmetric, non-linear distribution of variation across conductance states implies that programming to HCS will reduce the impact of variabilities on accuracy.

Digital Weight	Conductance (Variability)
00, 01	Low Conductance States (High Variability)
10, 11	High Conductance States (Low Variability)

Table 2.1: Differentiating between Low and High Conductance states based on variability in conductance states

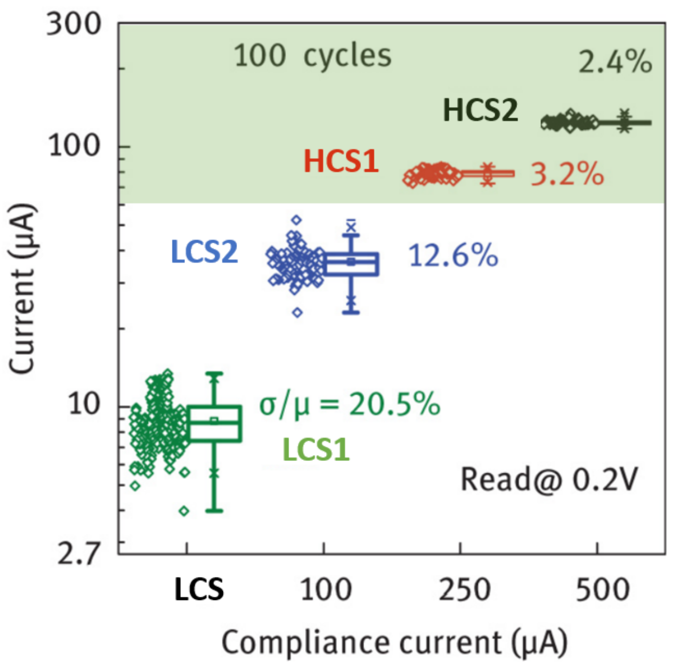


Figure 2.7: Observe the asymmetry in the conductance variation distribution over a 2 bit RRAM device. High Conductance State (HCS) exhibit lower conductance variability whereas LCS exhibit high variability in conductance [6]

It is important to leverage this variability to ensure that majority of RRAMs are mapped to High Conductance State (HCS). Hence, we outline the difference between Low and High Conductance States. This asymmetry is exploited to improve neural network accuracy on CIM inference engines.

2.4 Summary

- Fully Connected Networks (FCN) implement forward pass and backward propagation to assist neurons to learn classification tasks. Vector Matrix Multiplication (VMM) is the fundamental software computational operation. However on hardware, these are performed through Multiply-Accumulate (MAC) operations.

- Computation In-Memory (CIM) architectures facilitate energy and latency efficient VMM computation using traditional CMOS periphery (DAC & ADC).
- Memristors (NVM) are the fundamental devices that enable in-memory computation. Owing to their long retention, zero power leakage, multi-level capability, Resistive Random Access Memory (RRAM) devices are a promising candidate for scalable CIM frameworks.
- Conductance variability (C2C and D2D) reduces neural network accuracy. We recognized the asymmetry in conductance variability across low and high conductance states is an important factor to consider when looking to improve neural network accuracy.

CHAPTER 3

Related Works

This chapter provides a detailed review of related works for conductance mapping implementations on CIM architectures. Section 3.1 discusses the emergence of bit-slicing methods to map weights onto crossbar arrays. Next, Section 3.2 discusses works that look to extend the dynamic range in crossbar arrays using differential weight implementations. Section 3.3 discusses other strategies to improve accuracy by mitigating non-idealities concluding with a summary.

3.1 Bit Slicing CIM architectures

1-bit RRAM implementations require power-hungry periphery that would negate any energy gains presented by CIM architectures. Hence, neural weight mapping techniques are needed to reduce reliance on peripheral circuits. Considering the limited resolution of RRAMs to represent neural network weights, Shaifee *et. al.* [7] proposed a bit-slicing technique to decompose the weight bits into smaller bit slices. These bit sliced neural network weights can then be mapped onto an RRAM.

The ISAAC architecture proposed by Shaifee *et. al.* uses a 2 bit resolution 1-RRAM implementation to represent weights. It considers a 16 bit resolution for input voltages which is fed through the DAC 1 bit per timestep and an 16 bit resolution for the network weights. Additionally, they proposed a balanced bit slicing scheme and additional shift-add logic for accumulation of scaled partial current outputs. In balanced bit slicing, the 16 bit weights are mapped onto 8 RRAMs along the same WordLine (WL). This mapping mechanism can be seen in Figure 3.1.

Furthermore, in order to ensure accurate computation of Vector Matrix Multiplication (VMM) operations, they ensured to scale the currents based on the significance of the bit-slice. This is done to ensure that the accumulated partial digital output accurately represents the VMM computation.

Using Ohms Law and Kirchoff's law, the current output at the end of each BitLine (BL)

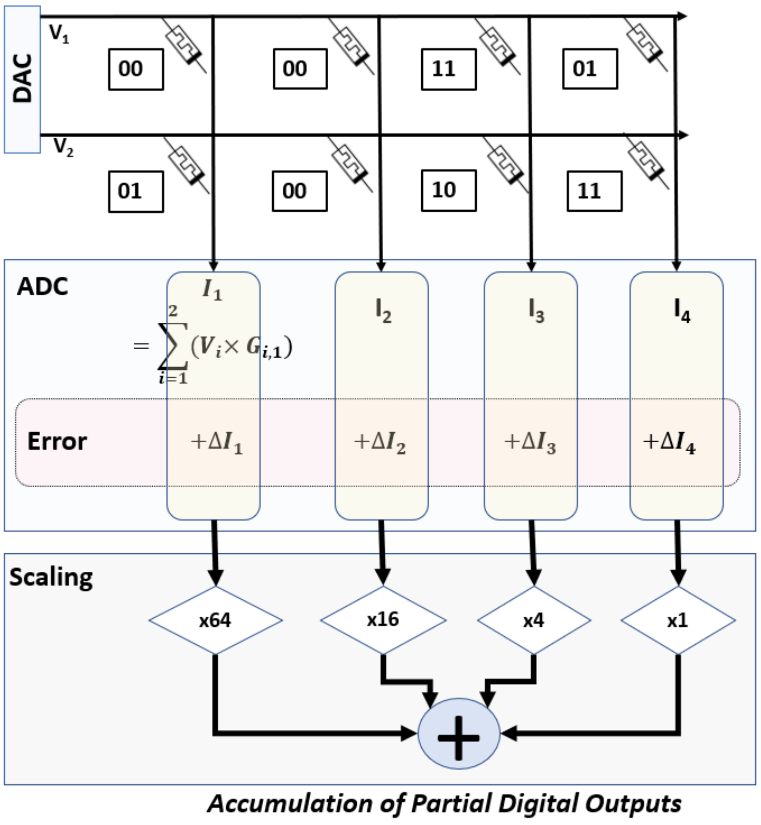


Figure 3.1: Instance of CIM architecture depicting the balanced bit slicing scheme. Here the 8 bit weights are mapped to 4 RRAM devices across the BitLine (BL) [7]

represents a partial output. Hence, the final digital output is represented as:

$$\text{Scaled Digital Output} = 64\cdot(I_1+\Delta I_1)+16\cdot(I_2+\Delta I_2)+4\cdot(I_3+\Delta I_3)+1\cdot(I_4+\Delta I_4) \quad (3.1)$$

However, Shaifee *et. al.* do not consider any RRAM non-idealities when performing experiments for the ISAAC framework using balanced bit-slicing schemes. Furthermore, no evaluation of neural network accuracy is presented and discussed to ascertain its reproducibility across various datasets. However, this research was critical to investigating the optimal parameters for the peripheral circuit to facilitate VMM computation. Extending this work, researchers [10, 56–58] have devised various bit-sliced CIM frameworks capable of implementing neural networks for recognition tasks.

3.2 Differential weight representations for CIM

The limited accuracy of bit-sliced implementations asked for a higher flexbility for programming conductances to achieve equivalent neural weight representation. Addionally, Bichler *et. al.* [12] recognized that 10x more energy was needed to transition from LCS to the HCS compared to transitioning from HCS to LCS. Hence, they proposed a differential weight representation as seen in Figure 3.2. A two-bit fixed point neural weight $W_{m,n}$ (m, n represents row and column on crossbar array) is represented as a different in conductance between two RRAMs. G_+ and G_- denote the positive and negative conductance contribution respectively.

A weight of '2' (bit 10) at row 'm' and column 'n' on the crossbar array is mapped as the difference between the conductance bit value '10' (G_+ RRAM contribution) and conductance bit value '00' (G_- RRAM contribution). Hence, the cumulative conductance is the difference between conductance of an RRAM for the positive contribution and RRAM conductance accounting for negative contributions to the neural weight. This provides the benefit of a higher dynamic representation for weight values.

$$W_{m,n} = G_+ - G_- \tag{3.2}$$

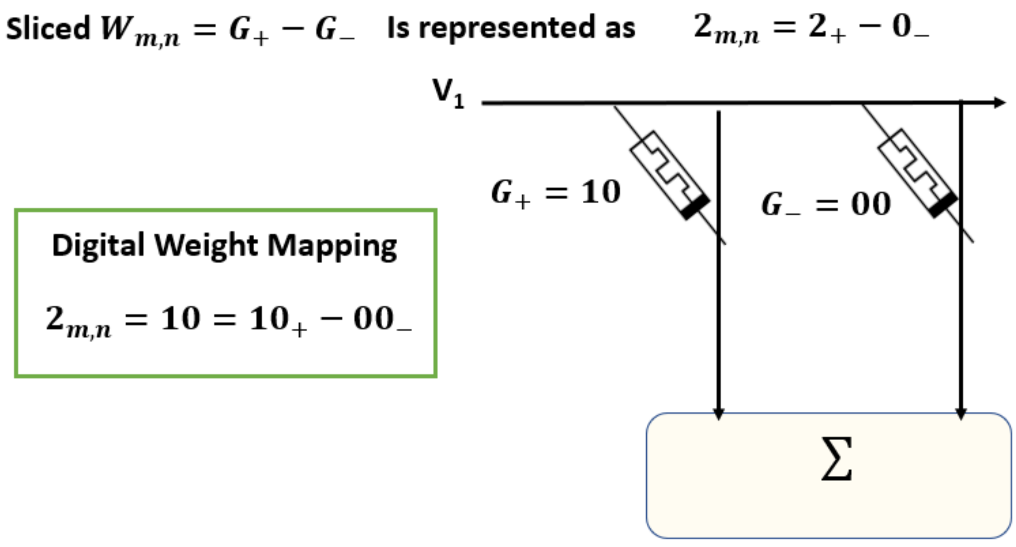

Sliced $W_{m,n} = G_+ - G_-$ Is represented as $2_{m,n} = 2_+ - 0_-$
V_1
$G_+ = 10$
$G_- = 00$
Digital Weight Mapping
$2_{m,n} = 10 = 10_+ - 00_-$
Σ

Figure 3.2: Depiction of differential weight representation using 2 RRAMs to represent neural weights using cumulative conductance

However, this method induces a high write cost associated with closed-loop programming. The contribution of differential implementations was extended by Gonugondla *et. al.* [48] who investigated different methods to reduce the write cost and improve accuracy of writing values to the crossbar array. Although their proposed approach lowers write cost by 5x - 10x, they report low accuracy in the presence of non-idealities.

To further investigate the capabilities of differential weight mapping representation on RRAMs, Burr *et. al.* [13] focused on exploring the positive conductance contribution ($G+$) and the negative RRAM component (G_-). They proposed the *Alternating Bidirectional* and *Fully Bidirectional* weight programming methods. In the alternating scheme, either G_+ or G_- is programmed whereas in the fully bidirectional scheme, both G_+ and G_- are simultaneously programmed to achieve the desired weight representation.

Burr *et. al.* [13] investigated the effective range of the differential RRAM configuration for programming weights. In the alternating bidirectional method, large weights are ignored since one/both RRAMs are in HCS or LCS.

On the other hand, the fully bidirectional programming method is effective in ensuring small conductances simultaneous programmed to G_+ and G_-. This ensures a wider range for weight representation. Joshi *et. al.* [59] implemented and validated the schemes, however both Burr *et. al.* [13] and Joshi *et. al.* [59] did not account for variabilities.

Li *et. al.* [8] investigated weight representation while accounting for conductance variabilities. They operated the CIM framework in Ex-Situ. During offline training, they downsampled the datasets to 8x8 pixels to reduce the number of RRAMs mapped to low conductance states. The main contribution by the researchers, was to tolerate conductance variability by programming the RRAM devices using only SET operations. Since RESET operations are expensive and more difficult control while performing conductance programming, only SET operations are performed to achieve the desired conductance.

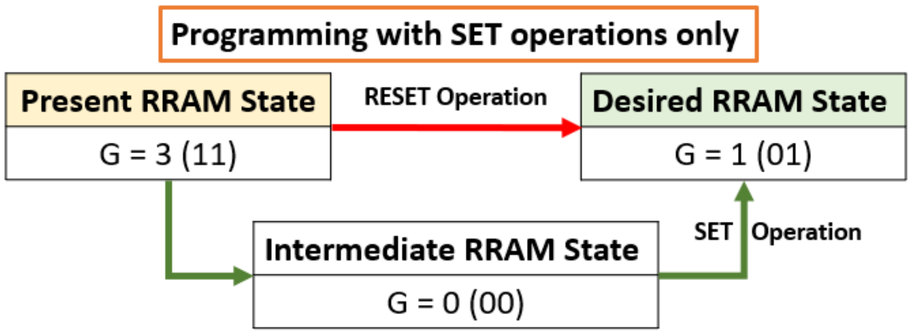


Figure 3.3: Programming RRAMs only using SET operations [8]

However, the researchers [8] observed that majority RRAMs still end up being programmed to low conductance (high variation) states. Furthermore, they assumed a narrow (small) variation profile.

These state-of-the-art conductance mapping techniques do not effectively address variability which significantly impacts Deep Neural Networks (DNN) accuracy.

3.3 Other relevant works

At a circuit and system level abstraction, significant research has been conducted to improve accuracy. Methods such as redundant arrays [36] have been proposed to overcome device failures on the crossbar array, but lead to unused resources.

An approach proposed by Jain *et. al.* [9] implements software (offline) retraining of neural networks to improve accuracy. This enables them to improve inference accuracy up to 26%

with 150 retraining iterations for ImageNet. The recognition inference accuracy saturates at ≈ 65% . However, Jain *et. al.* [9] relies significantly on algorithmic compensations that induce an area overhead and present CIM hardware design challenges.

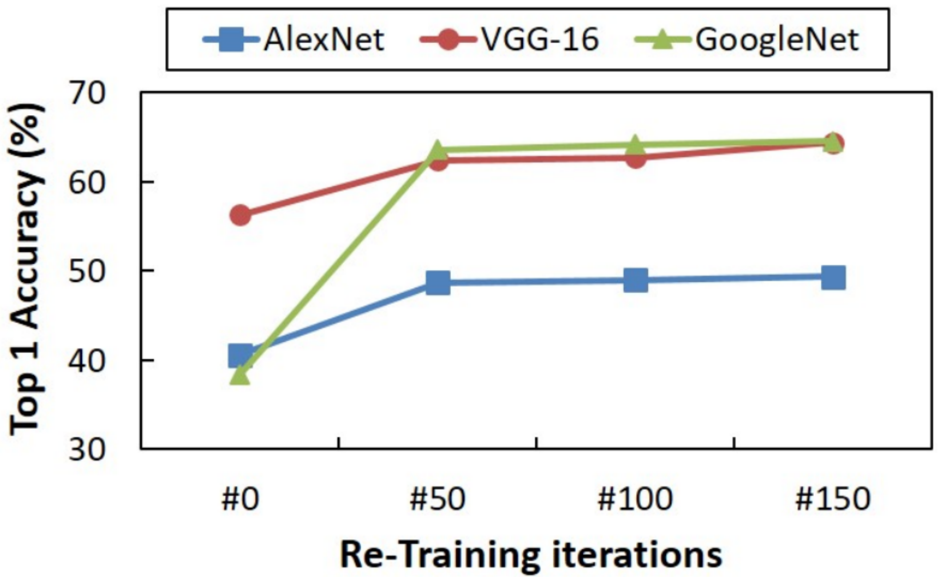


Figure 3.4: Software retraining improves inference accuracy but heavily relies on off-chip frameworks that have a large area and energy footprint [9]

Kim *et. al.* [60] proposed a HW-SW optimization to account for variabilities. The authors perform offline training with randomly generated weight defects to account for conductance variability with crossbar array redundancies. They observed that intentionally introducing random defects improves the robustness and accuracy of inference. However, this study only focuses on Device-to-Device (D2D) variabilities.

3.4 Summary

State-of-the-art implementations of Deep Neural Networks (DNN) on CIM architectures have been discussed. It is important to revisit common challenges that need to be further addressed.

- Bit-Sliced implementations [7, 10, 11, 56, 57] and Differential weight implementations [8, 12, 13] suffer from significant accuracy degradation across datasets. These conductance mapping techniques fail to effectively address impact of variability. Although some works address variabilities, they present high energy overheads and additional design challenges. Furthermore, these works do not leverage the asymmetry in variability exhibited by RRAMs.

- Other mitigation schemes [9, 36, 60–62] primarily rely on software intervention to model variabilities, retrain or prune networks to improve accuracy. However, they do not effectively address conductance variability leading to inadequate inference accuracy.

In this thesis we address the low accuracy of DNN due to conductance variability through mapping techniques that leverage the asymmetry in RRAM variability across conductance states.

Chapter 4

Proposed Schemes

This chapter presents the proposed mapping schemes. First, Section 4.1 outlines the overall implementation scheme to realize mapping implementations on CIM architectures. Next, Section 4.2 explains the proposed Dynamic Scaling Column (DSC) method. Finally, in Section 4.3, the proposed Complementary Conductance Matrix (CCM) is detailed concluding with a summary in Section 4.4

As discussed in previous chapters, RRAM-based CIM neural network architectures present a promising and scalable candidate for modern AI workloads. However, state-of-the-art conductance mapping schemes do not effectively mitigate variabilities, which significantly degrades inference accuracy.

We present conductance mapping schemes that leverage the asymmetric conductance variation distribution to ensure that majority of the RRAM devices are mapped to states with low variation (HCS) to improve accuracy.

4.1 Implementation Overview

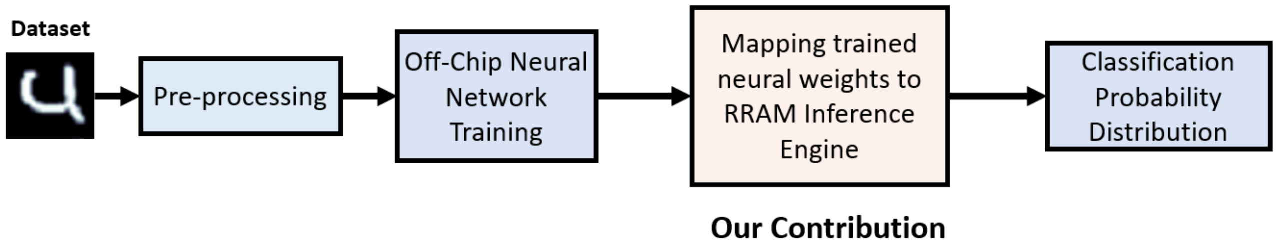

Dataset
Pre-processing
Off-Chip Neural Network Training
Mapping trained neural weights to RRAM Inference Engine
Classification Probability Distribution
Our Contribution

Figure 4.1: Implementation overview to realize neural network computation on CIM architectures.

Figure 4.1 outlines the implementation to realize the computation of VMM on CIM architectures. First, we preprocess and train the neural network off-chip to facilitate Floating-Point (FP) arithmetic needed for accurate weight convergence. Next, these optimal weights are extracted and converted to Fixed-point (FXP) representation. The input data is mapped to voltages through the DAC and weights are mapped to conductances. The two mapping proposals account for variability during inference. Finally, at the end of the forward-pass, the inference task is complete and produces a confusion matrix that validates the accuracy of the CIM neural network relative to the ground truth values.

4.2 Dynamic Scaling Column

The standard bit-slicing framework presented in ISAAC by Shaifee *et. al.* [7] is required for practical implementations for weight representation owing to the limited RRAM resolution. Furthermore, since partial current outputs need to be scaled according to bit-slice significance, the error in current is also scaled during the accumulation of partial digital outputs. This further reduces the accuracy of VMM operation performed in neural networks.

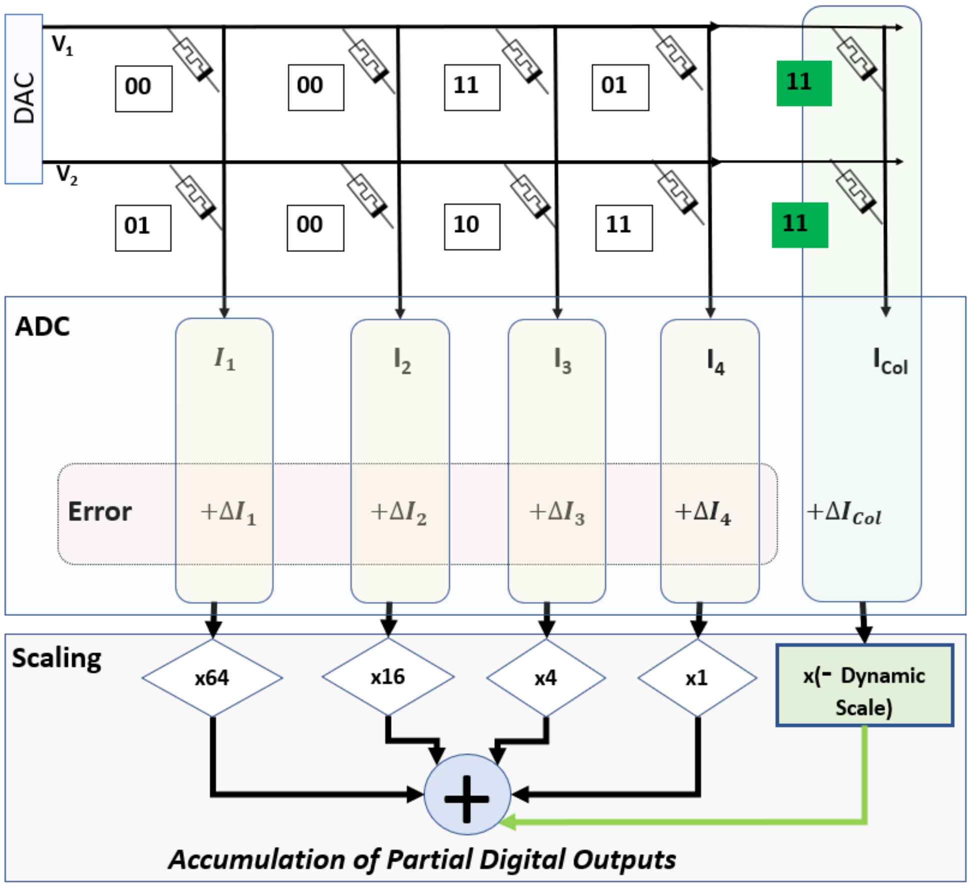


Figure 4.2: Introducing an extra column per weight (word) along with computing a dynamic scaling factor to suppress error of partial digital outputs

Previously, from the standard bit-sliced architecture presented by Shaifee *et. al.* [7], we observed the scaling of the current outputs to convert them to partial digital outputs. This scaling also amplified the errors across each BitLine (BL) as seen in Equation 3.1: $\text{Scaled Digital Output} = (64 \cdot (I_1 + \Delta I_1)) + (16 \cdot (I_2 + \Delta I_2)) + (4 \cdot (I_3 + \Delta I_3)) + (1 \cdot (I_4 + \Delta I_4))$

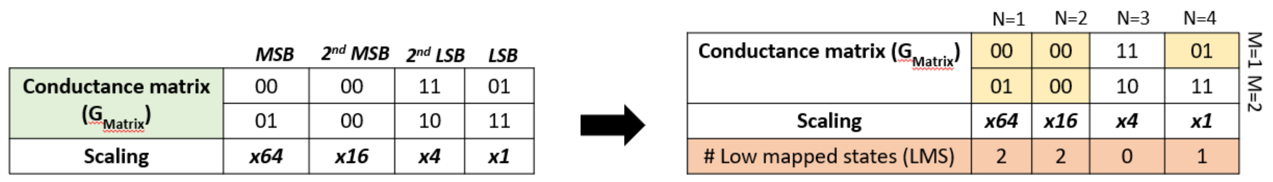
In order to reduce this error we propose the Dynamic Scaling Column (DSC). This proposal looks to suppress errors in the accumulation of partial current outputs by introducing an extra column of RRAMs and computing the Dynamic Scaling Factor (DSF). The architectural modification is shown in Figure 4.2. We introduce an extra column of RRAMs per neural weight which is always programmed to bit '11' (least conductance variability). This reduces the impact of ΔI_{col}. Next, we compute the optimal Dynamic Scaling Factor (DSF) to suppress partial digital errors to ensure the output of the VMM is close to the expected output. The effect of the Dynamic Scaling Column (DSC) can be represented mathematically as:

$$\text{Suppressed Digital Output} = \textit{Scaled Digital Output} - (\mathbf{DSF} \cdot (I_{col} + \Delta I_{col})$$

This method reduces the computation error when accumulating the partial digital outputs. In order to find the optimal Dynamic Scaling Factor (DSF), we implement Algorithm 1.

The algorithm to compute the Dynamic Scaling Factor (DSF) involves three steps.

Step 1: First we retrieve the original conductance matrix and track the number of low mapped states per scaling factor as seen in Figure 4.3. This provides an indication regarding the potential impact of the variability due to low mapped states on the final MAC operation.



	MSB	*2nd MSB*	*2nd LSB*	*LSB*
Conductance matrix (G_{Matrix})	00	00	11	01
	01	00	10	11
Scaling	***x64***	***x16***	***x4***	***x1***

	N=1	N=2	N=3	N=4	
Conductance matrix (G_{Matrix})	00	00	11	01	M=1
	01	00	10	11	M=2
Scaling	***x64***	***x16***	***x4***	***x1***	
# Low mapped states (LMS)	2	2	0	1	

Figure 4.3: Step 1 of proposed DSC: Tracking number of low mapped states (LMS)

Step 2: This involves computing the Dynamic Scaling Factor (DSF) based on the number of low mapped states (LMS) as well as the corresponding scaling factor. It is important to recognize that errors in current outputs due to variability will be much higher for RRAMs programmed to LCS. Hence, we develop a heuristic to compute the ideal DSF based on which scaling factor contribution contains the most low mapped states across the BitLine (BL).

Algorithm 1 Computing Dynamic Scaling Factor (DSF)

```
Input: $G_{matrix} = G_{M,N}$ and $|G_{matrix}| = M \cdot N$        ▷ Original Conductance Matrix
Output: DSF and $G_{DSF} = G_{M,N+1}$        ▷ Dynamic Scaling Factor
  Step 1
  for $G_{matrix}$ do
      $\sum_{n=1}^{N} LMS \leftarrow |G_{m,n} == \text{LCS}|$        ▷ Track low mapped states → LMS
  end for
  Step 2
  if $\sum_{n=1}^{N} LMS < \frac{M \cdot N}{2}$ then
      $DSF = -LSB$        ▷ Condition 1
  else if $\sum_{n=1}^{N} LMS >= \frac{M \cdot N}{2}$ then
      if $\sum_{n=1}^{\frac{N}{2}} LMS < \sum_{n=\frac{N}{2}+1}^{N} LMS$ then
          $DSF \leftarrow$ -2nd LSB        ▷ Condition 2A
      else if $\sum_{n=1}^{\frac{N}{2}} LMS >= \sum_{n=\frac{N}{2}+1}^{N} LMS$ then
          $DSF \leftarrow$ - MSB        ▷ Condition 2B
      end if
  end if
  Step 3
  $G_{DSF} \leftarrow G_{M,N} + ((Gmax_{M,1} \cdot DSF)$        ▷ Extra column scaled with DSF
```

- Condition 1: When the number of low mapped states is less than half the total elements (not a sparse matrix) in the conductance matrix, we set the Dynamic Scaling Factor (DSF) to a negative scaling corresponding to the LSB, *i.e.* -2^0. This is mathematically expressed below

$$\sum_{n=1}^{N} LMS < \frac{M \cdot N}{2}, \text{DSF} = -LSB$$

- Condition 2: When the number of low mapped states is greater than or equal to half the number of elements in the conductance matrix (is a sparse matrix), we recognize that majority RRAMs are mapped to LCS and hence will have a significant impact on errors when accumulating current outputs.

$$\sum_{n=1}^{N} LMS >= \frac{M \cdot N}{2}$$

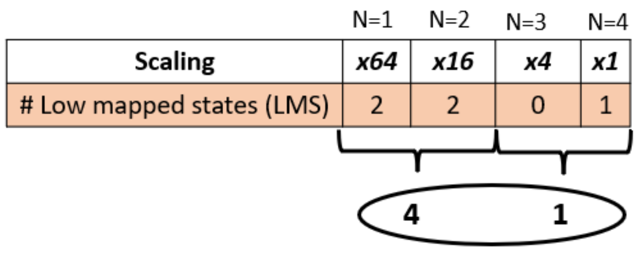
We analyze two situations to compute the DSF.

A: Based on the accumulation scaling factor, when the sum of the number of low mapped states for MSB and 2nd MSB is less than the sum of the number of low mapped states in 2nd LSB and LSB. For this condition, the DSF is set to the negative scaling contribution corresponding to the 2nd LSB, *i.e.* -2^4. This is expressed in mathematical notation as below.

$$\sum_{n=1}^{\frac{N}{2}} LMS < \sum_{n=\frac{N}{2}+1}^{N} LMS; \text{DSF} = -2^{nd} LSB$$

B: When the sum of the number of low mapped states for MSB and 2nd MSB is greater than or equal to the sum of the number of low mapped states in 2nd LSB and LSB, we expect partial output errors to be maximum. Hence, we set the DSF to the negative scaling contribution corresponding to MSB, *i.e.* -2^6.
MSB and 2nd MSB is less than the sum of the number of low mapped states in 2nd LSB and LSB. This can be translated into a mathematical expression.

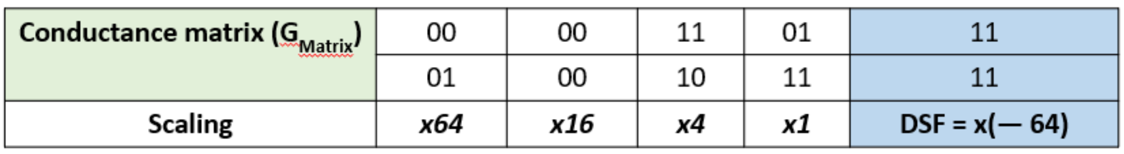
$$\sum_{n=1}^{\frac{N}{2}} LMS >= \sum_{n=\frac{N}{2}+1}^{N} LMS; \text{DSF} = -MSB$$



	N=1	N=2	N=3	N=4
Scaling	*x64*	*x16*	*x4*	*x1*
# Low mapped states (LMS)	2	2	0	1

Figure 4.4: Step 2 of proposed DSC: Compute Dynamic Scaling Factor (DSF) based on scaling

Step 3: In this step, we introduce an extra column of '11' mapped RRAMs and scale the bitline with the computed Dynamic Scaling Factor (DSF). Figure 4.5 shows how the Dynamic Scaling Column (DSC) facilitates the suppression of errors during accumulation of partial currents.



Conductance matrix (G_{Matrix})	00	00	11	01	11
	01	00	10	11	11
Scaling	***x64***	***x16***	***x4***	***x1***	**DSF = x(— 64)**

Figure 4.5: Step 3 of proposed DSC: Add an extra column of RRAMs and scale them by the Dynamic Scaling Factor (DSF).

4.3 Complementary Conductance Matrix

The Complementary Conductance Matrix (CCM) proposal is a hardware-software co-optimization that looks to preserve the original contents of the weight matrix, while also ensuring that majority RRAMs are mapped to HCS. This will leverage the low variability in HCS and reduce the errors produced in the Vector Matrix Multiplication (VMM) result. Figure 4.6 shows the overview of realizing this proposed scheme.

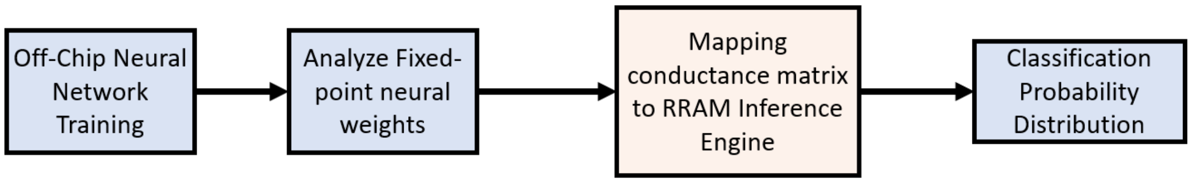

Off-Chip Neural Network Training
Analyze Fixed-point neural weights
Mapping conductance matrix to RRAM Inference Engine
Classification Probability Distribution

Figure 4.6: Overview of steps involved to realize Complementary Conductance Matrix (CCM) proposed scheme

Once the neural network has been trained offline, the optimal floating point weights are converted to fixed point representation. Next, the neural weights are sliced using the standard bit-slicing scheme. Since now there is prior knowledge of values that need to be programmed to the RRAM, we analyze whether the majority of RRAMs will be mapped to low conductance states. This implies that most neural weights are small. Based on this information, we extract the G_{max}-complement matrix to perform VMM using a differential weight representation. This ensures that we preserve the original neural weight representation while ensuring majority of the devices are mapped to high conductance states. This implementation can be realized in the following manner.

Step 1: Once the optimal Floating-Point (FP) weights after Ex-Situ operation is converted to Fixed-point (FXP) representation, the complementary matrix of the original conductance matrix is extract. This is done by taking the bit slice complement of the G_{max}, '11' state.

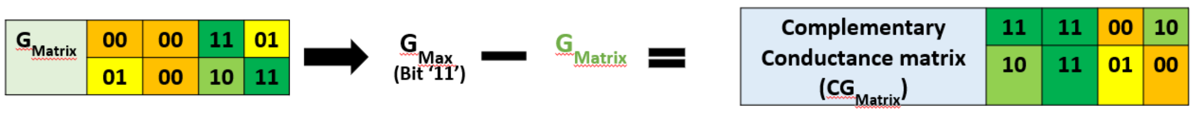


Figure 4.7: Step 1 in proposed CCM: Extract the Complementary Conductance Matrix (CCM).

Step 2: Using differential weight representation, we perform Vector Matrix Multiplication (VMM) while preserving the weights represented in the original conductance matrix G_{Matrix}.

$$I = V \times [G_{Max} - CG_{Matrix}]$$

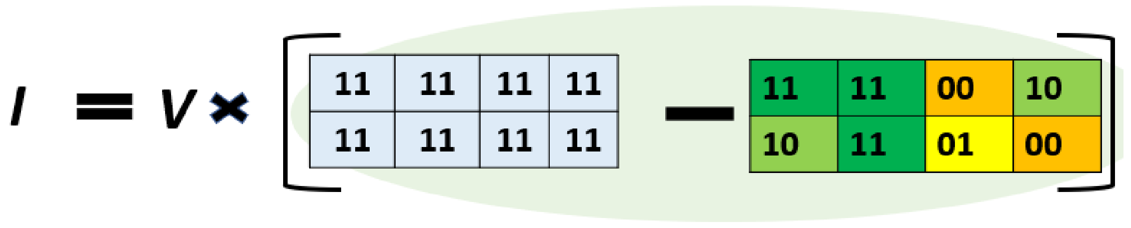


Figure 4.8: Step 2 in proposed CCM: Map matrix on a differential crossbar

This ensures that majority RRAMs are mapped to HCS reducing the impact of variability on VMM output.

4.4 Summary

This chapter introduces and discusses the proposed schemes to realize high accuracy Deep Neural Networks (DNN) inference engines on CIM architectures.

- We propose two conductance mapping schemes that mitigate conductance variation of RRAMs on CIM architectures for neural networks. We perform Ex-Situ training to extract optimal neural weights. These weights are mapped onto the CIM framework to perform DNN inference.
- **Dynamic Scaling Column:** We introduce an extra column of high '11' state RRAMs to reduce the error when accumulation of partial current outputs at the end of each BitLine (BL). The error due to the extra column is negligible since all RRAMs in the extra column are programmed to High Conductance State (HCS), which exhibit low variability.
- **Complementary Conductance Matrix:** The G_{max}-complement of the original Fixed-point (FXP) weight representation is extracted and mapped onto the crossbar array. This minimizes the number of RRAMs mapped to Low Conductance State (LCS). Using a differential mapping method, we preserve the original optimal Fixed-point (FXP) weights, while reducing the Vector Matrix Multiplication (VMM) computation errors due to variability.

CHAPTER 5

Simulation Results

This chapter presents the results of the proposed mapping schemes. Section 5.1 begins by detailing the datasets used and Section 5.2 outlines the software and hardware setup. We go on to define the metrics that will be used to evaluate the performance of the proposed methods in Section 5.3. Section 5.4 presents the accuracy for Dynamic Scaling Column (DSC), Complementary Conductance Matrix (CCM) and the combined implementation compared to state-of-the-art implementations. Further, we discuss the area and energy overheads induced by the proposed methods and conclude by aggregating the observed results to evaluate trade-offs in order to reach a design decision.

5.1 Datasets

All datasets are converted to 28x28 pixel gray-scale image, where each pixel value ranges from 0 to 255 indicating the brightness of the pixel. Furthermore, all datasets have 60,000 training images and 10,000 test images.

MNIST [63]: The MNIST (Modified National Institue of Standards and Technology) database contains hardwritten digits from 0 to 9.

Fashion MNIST (FMNIST) [64]: The Fashion-MNIST database is a catalogue created by Zalando consisting of articles of clothing separated into 10 classes.

Kuzushiji MNIST (KMNIST) [65]: KMNIST is a drop in replacement for MNIST but contains Japanese characters that require to be classified into 10 categories.

Rotated MNIST (MNIST-rotDIG): This flavor of the MNIST dataset involves implementing small rotational perturbations to the original dataset.

Extended MNIST (EMNIST) [66]: This dataset includes 10 balanced classes of digits, uppercase and lowercase handwritten alphabets.This is a more challenging classification task compared to MNIST.

5.2 Simulation Setup

5.2.1 Software Setup

Deep Neural Networks (DNN)s are implemented using an open-source Python library called PyTorch [67]. This facilities rapid production of neural networks suited to prediction tasks. PyTorch provides dataloader utilities to easily feed dataset to the DNN for training or inference. This framework allows hyperparameters to be controlled to improve learning as well as helps extract Floating-Point (FP) weights from a trained neural network as a tensor. Numpy is used to perform the central VMM operation. The NumPy library in Python is used to perform bit slicing and differential implementations. Finally, we implement element-wise multiplication operations to produce the final result of the VMM.

Deep Learning Parameters	
Network	Fully Connected (FCN)
No. of hidden layers	2
Neurons	$784 \longrightarrow 100 \longrightarrow 50 \longrightarrow 10$
Learning Algorithm	Stochastic Gradient Descent
Learning Rate	$1e-3$
Training	Software: Floating Point

Table 5.1: The deep learning network and parameters adopted for this thesis.

This thesis implements a Fully Connected Networks (FCN) with 2 hidden layers. The input layer contains 784 neurons, the first hidden layer has 100 neurons, the second hidden layer has 50 neurons. Finally, the output classification has 10 neurons to generate the confusion matrix indicating the probability distribution of the neural network predictions. We perform the training in software to obtain Floating-Point (FP) neural weights. The training uses stochastic gradient descent (SGD) as the learning algorithm and a learning rate of $1e-3$.

The software inference accuracy for various datasets is shown in Table 5.2.

Dataset	Software inference accuracy
MNIST [63]	97%
Fashion-MNIST [64]	88%
K-MNIST [65]	88%
Rotated MNIST	97%
Extended-MNIST [66]	82%

Table 5.2: Software inference accuracy across five datasets

5.2.2 Hardware Setup

The CIM architecture implemented in this thesis follows the fundamental framework provided by Shaifee *et. al.* [7] in the ISAAC architecture. The crossbar array size is 128x128. For inference, the inputs are 16 bit fixed point representations whereas the weights are 16 bit fixed point values. We model the conductance variation according to the device characteristics presented by Prakash *et. al.* [6]. Furthermore, we assume a HCS/LCS ratio of 20.

RRAM Crossbar Hardware Parameters	
Crossbar Size	128 x 128
Inference	Inputs: 16-bit fixed point Weights: 16-bit fixed point
Max. Conductance G_{max}	500 μS
Ideal Conductance Profile (μS)	[0, 166.7, 333.3, 500]
Variation Profile (σ/μ) % [6]	[20.5, 12.6, 3.2, 2.4]
Conductance On/Off Ratio	20

Table 5.3: The CIM setup inline with the implementation presented by Shaifee *et. al.* [7].

The energy and area footprint of the DNN inference engine is estimated based on the ISAAC architecture [7].

Components	**Energy (pJ)**	**Area (mm^2)**
Input Register	124	0.0021
Output Register	23	0.00078
RRAM (128x128)	240	0.0002
DAC	400	0.00017
ADC	1600	0.0096
Sample & Hold	0.00097	0.00004
Shift & Add	20	0.00024

Table 5.4: Energy and Area Consumption of an IMA CIM components derived from Shaifee *et. al.* [7]

5.3 Performance Metrics

We use the following metrics to demonstrate the effectiveness of our contributions in this thesis:

- **Raw Accuracy (%):** It is the accuracy achieved by traditional software implementations using CPUs and GPUs.
- **Relative Accuracy (%):** It is the ratio between the accuracy obtained on a fixed point crossbar relative to the raw accuracy. It indicates the accuracy recovered by CIM

hardware compared to software. Relative Accuracy:

$$\frac{\text{Crossbar fixed point accuracy}}{\text{Raw Accuracy}} \times 100$$

- **Energy (pJ):** It is the estimated total on-chip IMA energy consumption of the CIM inference engine while performing VMM computation.
- **Area (mm^2):** This is the estimated IMA area consumption of the CIM inference engine including peripheral circuits.
- **Figure-Of-Merit ($GOPs/W$):** Accuracy $\times$ Energy Efficiency, where energy efficiency is expressed in *giga operations per second per watt*.

5.4 Simulation Results

5.4.1 Classification accuracy evaluation

5.4.1.1 Dynamic Scaling Column

Figure 5.1 shows the relative accuracy of the proposed Dynamic Scaling Column (DSC) compared to bit-sliced and differential implementations across all datasets. An accuracy improvement up to 2.85x is observed compared to bit-sliced implementations [7, 10, 11]. Additionally, across all datasets, an average of 30% accuracy improvement was obtained compared to differential weight representations [8, 12, 13].

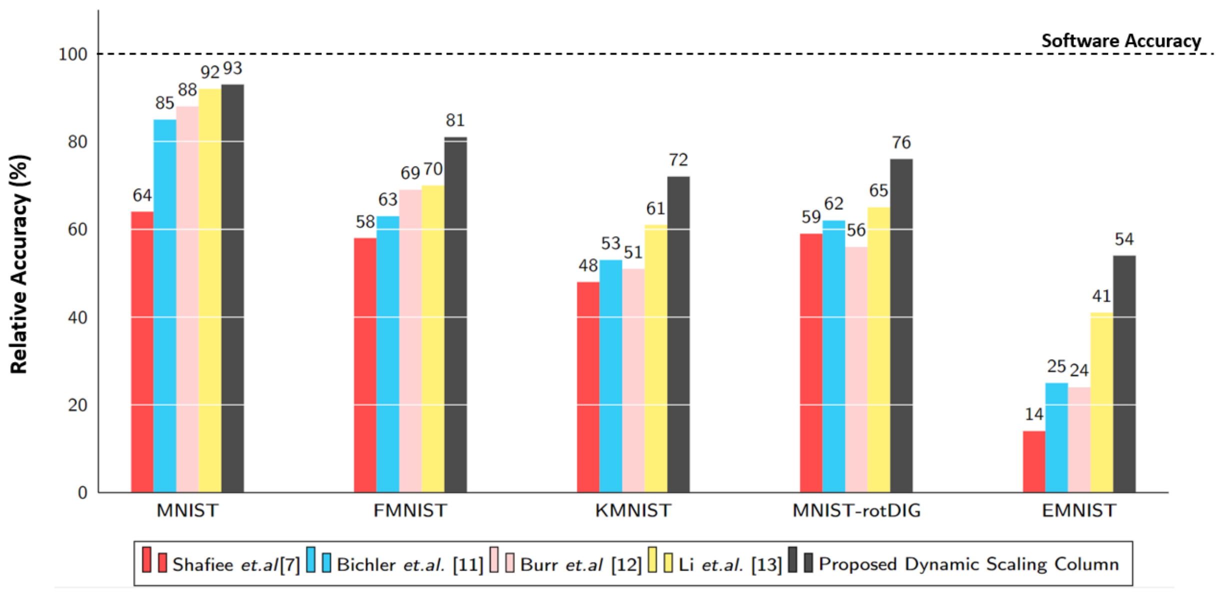


Figure 5.1: Relative accuracy comparison across five datasets for bit-sliced [7, 10, 11], differential [8, 12, 13] and the proposed Dynamic Scaling Column (DSC) mapping implementation.

The extra column of RRAMs mapped to bit '11' state along with the heuristic computation of the Dynamic Scaling Factor (DSF) reduces the error in accumulation of partial current outputs. The variability due to the extra RRAMs is negligible since all devices are mapped to High Conductance State (HCS). Suppression of errors is dynamic over different conductances mapped onto the crossbar

5.4.1.2 Complementary Conductance Matrix

In the proposed Complementary Conductance Matrix (CCM), the original mapped weights are manipulated to ensure majority RRAMs are programmed to a HCS. By taking the G_{max} complement of the original conductance matrix, G_{Matrix}, the complementary matrix is mapped onto a differential crossbar array. Since HCS exhibit low conductance variability and majority RRAMs are mapped to HCS, a significant improvement in accuracy is observed.

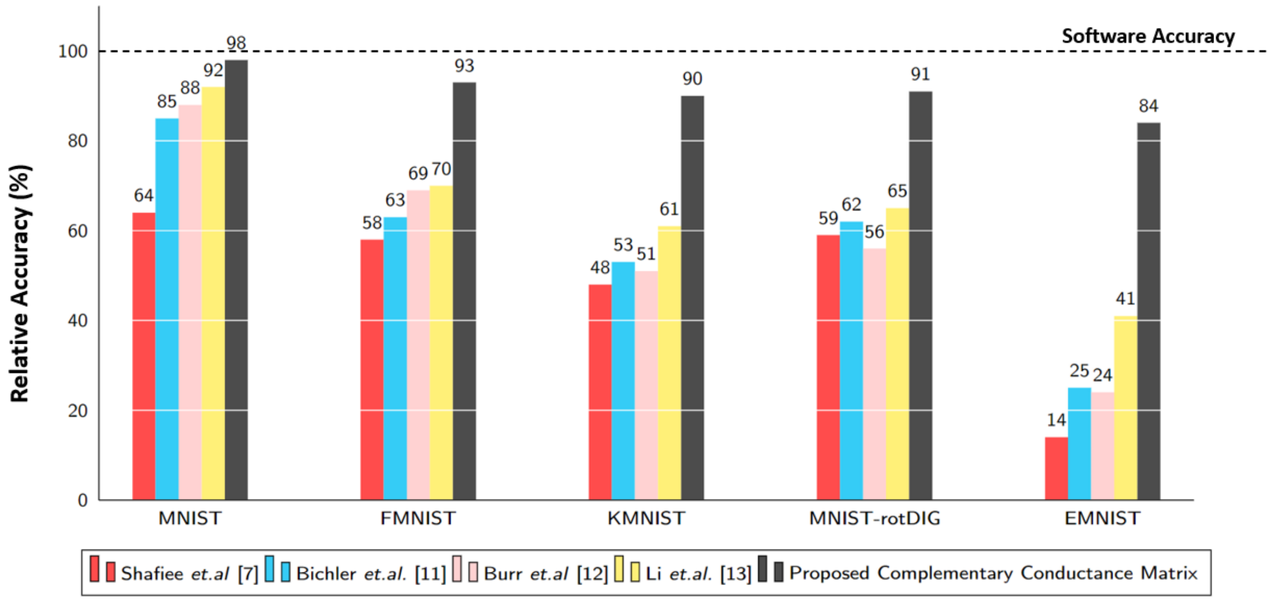


Figure 5.2: Relative accuracy comparison across five datasets for bit-sliced [7, 10, 11], differential [8, 12, 13] and the proposed CCM mapping implementation.

The proposed Complementary Conductance Matrix (CCM) method reports up to 5x accuracy improvement compared to bit-sliced implementations [7, 10, 11]. Across all datasets, an average of 1.5x improvement in accuracy over differential implementations [8, 12, 13] is obtained. Furthermore, this method does not induce any area or energy overhead compared to differential implementations but uses 2x the number of RRAMs which can be attributed to the differential weight representation of the proposed CCM on the crossbar array.

Complementary Conductance Matrix (CCM) obtains accuracy improvements since all neural weights are biased towards High Conductance State (HCS). These states exhibit the least conductance variation. This provides accurate weight representation while minimizing the impact of conductance variations. Furthermore, programming the positive contribution (G_+)of RRAMs only to HCS reduces the complexity of the write mechanism. Hence, implementing the Complementary Conductance Matrix (CCM) provides significant benefits for neural networks.

5.4.1.3 Combined proposal - DSC & CCM

Combining the DSC and CCM proposals reports the best accuracy improvements. This significantly outperforms all state-of-the-art techniques across all datasets. Comparing the accuracy for E-MNIST (most difficult task), we observe an improvement of up to 5.43x compared to bit-sliced [7, 10, 11] implementations. Furthermore, compared to differential implementations [8, 12, 13], a 2.6x improvement in accuracy is recorded

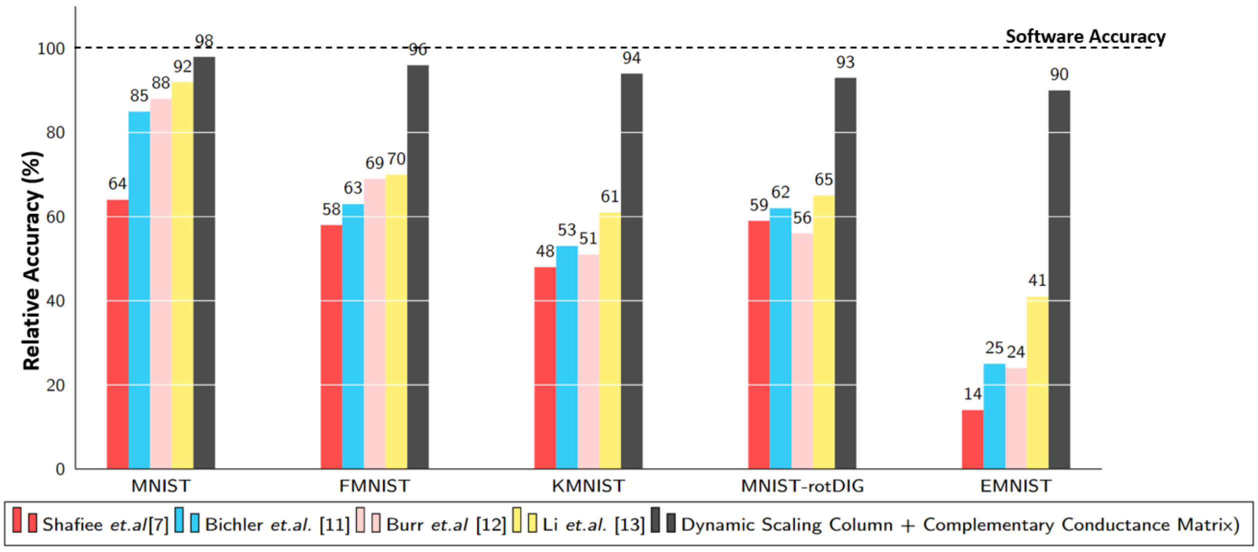


Figure 5.3: Relative accuracy comparison across five datasets for bit-sliced [7, 10, 11], differential [8, 12, 13] and the combined methodology (proposed DSC + proposed CCM)

This can be attributed to the fact that the Dynamic Scaling Column (DSC) method suppresses variability in partial outputs by scaling the '11' mapped RRAM column with the Dynamic Scaling Factor (DSF). Additionally, by extracting the Complementary Conductance Matrix (CCM), we reduce the number of RRAMs mapped to LCS. Hence, we reduce the conductance variation across the crossbar array.

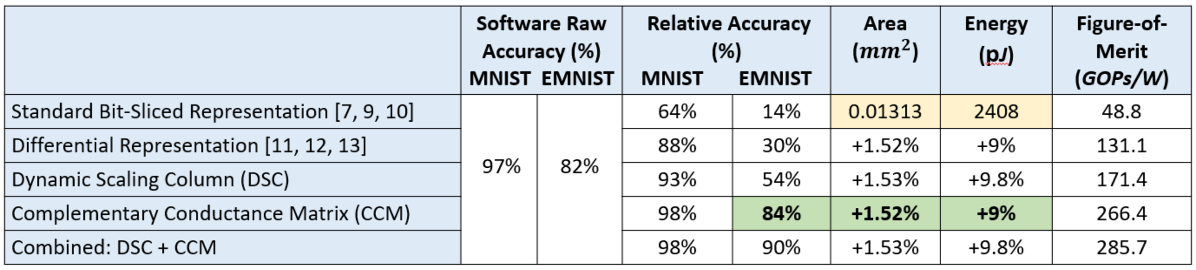
5.4.2 Performance Analysis with state-of-the-art

The bit-sliced implementation consumes 2408pJ of energy and 0.01313mm^2 of area per IMA. This energy consumption is dominated by ADCs. The proposed methodologies induce a minimal area overhead compared to the baseline bit-sliced implementation. The proposed Dynamic Scaling Column (DSC) method induces a 1.53% overhead in area compared to bit-sliced implementation due to the extra column and differential weight implementation. This area overhead of the extra column is negligible and regarding the periphery since ADC resources are shared, the ADC will require more energy to complete the Multiply-Accumulate (MAC) operations. Furthermore, since DSC uses differential weight representation [8, 12, 13], compared to bit-sliced implementations (baseline) [7, 10, 11], double the number of RRAMs are used. However, RRAMs do not significantly contribute to the overall area consumption of the chip.

The Complementary Conductance Matrix (CCM) method induces no area overhead relative to differential implementations since only RRAM mapped values are manipulated but incurs a 1.52% area overhead compared to bit-sliced implementations (baseline). The combined proposal increases the on-chip footprint by $2.02{\cdot}10^{-4}mm^2$ compared to bit-sliced implementations due to the extra column of RRAMs and the differential weight representation [8, 12, 13].

In DSC, we observe a 9.8% increase in on-chip energy due to its differential implementation and ADC sharing resources to facilitate the extra column compared to bit-sliced implementations. CCM uses same energy as differential representations and so incurs a 9% cost compared to standard bit-sliced implementations. Finally, with respect to energy consumption of the combined method, since architecture of RRAM and ADC elements are similar it consumes on-chip energy equivalent to Dynamic Scaling Column (DSC) proposed implementation.

We collate all the results presented on accuracy, energy, area and figure-of-merit. The proposed Dynamic Scaling Column (DSC), Complementary Conductance Matrix (CCM) and combined method significantly outperform all state-of-the-art mapping implementations across all datsets. The least improvement is seen across the simple MNIST dataset since state-of-the-art bit-sliced and differential implementations are already capable of competing with traditional software accuracy. However, towards the more difficult dataset, Extended-MNIST, the largest accuracy gains compared to state-of-the-art is observed.



	Software Raw Accuracy (%)		Relative Accuracy (%)		Area (mm^2)	Energy (pJ)	Figure-of-Merit (*GOPs/W*)
	MNIST	EMNIST	MNIST	EMNIST			
Standard Bit-Sliced Representation [7, 9, 10]	97%	82%	64%	14%	0.01313	2408	48.8
Differential Representation [11, 12, 13]			88%	30%	+1.52%	+9%	131.1
Dynamic Scaling Column (DSC)			93%	54%	+1.53%	+9.8%	171.4
Complementary Conductance Matrix (CCM)			98%	**84%**	**+1.52%**	**+9%**	266.4
Combined: DSC + CCM			98%	90%	+1.53%	+9.8%	285.7

Figure 5.4: Holistic perspective of Bit-Sliced [7, 10, 11], differential [8, 12, 13], proposed DSC, CCM and combined method. CCM uses 1.52% more area and 9% energy compared to the standard bit-sliced representation

Although the DSC method improves accuracy up to 2.85x, the low accuracy in difficult tasks reduce the viability of this solution. On the other hand, the CCM proposal improves accuracy by 5x on the E-MNIST dataset compared to bit-sliced implementations but only incurs an area overhead of 1.52% and energy cost of 9% compared to bit-sliced implementations. Moreover, it is important to note that CCM does not incur any cost compared to differential implementations. Finally, the combined proposal reports an accuracy improvement of up to 5.4x but incurs a 9.8% energy overhead and 1.53% area overhead compared to bit-sliced implementations 7% on the E-MNIST dataset.

Looking at the trade-offs presented by the implementations, the proposed Complementary Conductance Matrix (CCM) mapping method is the most viable candidate for large scale CIM architectures for DNN inference engines.

Chapter 6

Conclusion

This chapter concludes this thesis. The work presented in this thesis is summarized. Lastly, we make recommendations for future research.

Conclusion

In this thesis, two conductance mapping schemes leveraging RRAM variability are proposed targeting a high accuracy bias-mapped CIM Deep Neural Networks (DNN) inference engine. RRAMs suffer from Cycle-to-Cycle (C2C) and Device-to-Device (D2D) variations that significantly degrade accuracy of neural networks. State-of-the-art bit-sliced and differential weight representation, enabled neural weight mapping on CIM frameworks however they lack focus on the impact of variabilities on accuracy. RRAM exhibit an asymmetric variability profile. High Conductance State (HCS) exhibit low variability. To reduce the error in VMM computation, it is important to leverage this low variability in order to minimize number of RRAM devices mapped to Low Conductance State (LCS). These observations motivated the proposed mapping schemes.

The Dynamic Scaling Column (DSC) suppresses accumulation errors during partial output computation. Across all datasets, DSC obtains accuracy improvements of up to 2.85x incurring a 1.54% area cost and a 9.8% energy overhead compared to (baseline) bit-sliced implementations. The Complementary Conductance Matrix (CCM) extract the G_{max} complement to ensure majority of the RRAMs are mapped to High Conductance State (HCS). CCM reports a 5x accuracy improvement incurring a 1.52% area cost and 9% energy overhead compared to bit-sliced implementations. However, with respect to differential implementations, an accuracy improvement of up to 2.36x is achieved at no area or energy cost.

Overall, the proposed schemes improved accuracy significantly across all datasets with a negligible overhead. However, based on accuracy-energy trade-offs, CCM is the preferred conductance mapping scheme to implement in RRAM-based CIM inference engine architectures for Deep Neural Networks (DNN).

Future Work

The thesis presents suggestions that can be adopted in future research:

- **Non-idealities** from a device, architecture and system level affect performance of neural network applications. CIM frameworks should integrate these non-idealities and define relationships across all levels. These frameworks will provide a more realistic perspective of CIM architectures

- **Binary Neural Network** implementations on CIM crossbar array. This can enable on-chip training and since neural weights are bounded, weight updates can be performed. Accuracy on easy datasets such as MNIST have been demonstrated. However, more complicate classification tasks are needed to investigate its true potential for real-world implementations.

Bibliography

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*.
San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2011.

[2] S. Jain, A. Ankit, I. Chakraborty, T. Gokmen, M. Rasch, W. Haensch, K. Roy, and A. Raghunathan, "Neural network accelerator design with resistive crossbars: Opportunities and challenges," *IBM Journal of Research and Development*, vol. 63, no. 6, pp. 10–1, 2019.

[3] ComputingCell, "Memristor technology getting closer," 2019.

[4] H. Aziza, H. Ayari, S. Onkaraiah, M. Moreau, J. M. Portal, and M. Bocquet, "Multilevel operation in oxide based resistive RAM with SET voltage modulation," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pp. 1–5, IEEE, 2016.

[5] H. Aziza, A. Perez, and J.-M. Portal, "Resistive RAMs as analog trimming elements," *Solid-State Electronics*, vol. 142, pp. 52–55, 2018.

[6] A. Prakash and H. Hwang, "Multilevel cell storage and resistance variability in resistive random access memory," *Physical Sciences Reviews*, vol. 1, no. 6, 2016.

[7] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with in-Situ Analog Arithmetic in Crossbars," *SIGARCH Comput. Archit. News*, vol. 44, p. 14–26, June 2016.

[8] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature communications*, vol. 9, no. 1, pp. 1–8, 2018.

[9] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "RxNN: A framework for evaluating deep neural networks on resistive crossbars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[10] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A Programmable Ultra-Efficient Memristor-Based Accelerator for Machine Learning Inference," ASPLOS '19, (New York, NY, USA), p. 715–731, Association for Computing Machinery, 2019.

[11] A. Ranjan, S. Jain, J. R. Stevens, D. Das, B. Kaul, and A. Raghunathan, "X-MANN: A crossbar based architecture for memory augmented neural networks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[12] O. Bichler, M. Suri, D. Querlioz, D. Vuillaume, B. DeSalvo, and C. Gamrat, "Visual pattern extraction using energy-efficient "2-PCM synapse" neuromorphic architecture,"

IEEE Transactions on Electron Devices, vol. 59, no. 8, pp. 2206–2214, 2012.

[13] G. W. Burr, R. M. Shelby, S. Sidler, C. di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. N. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

[14] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent ram," *IEEE Micro*, vol. 17, p. 34–44, mar 1997.

[15] R. Rizk, D. Rizk, A. Kumar, and M. Bayoumi, "Demystifying emerging nonvolatile memory technologies: understanding advantages, challenges, trends, and novel applications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2019.

[16] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "Cpu db: Recording microprocessor history," *Commun. ACM*, vol. 55, p. 55–63, apr 2012.

[17] Z. Abbas and M. Olivieri, "Impact of technology scaling on leakage power in nano-scale bulk cmos digital standard cells," *Microelectron. J.*, vol. 45, p. 179–195, feb 2014.

[18] S. Mittal, "A Survey of ReRAM-Based Architectures for Processing-In-Memory and Neural Networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 75–114, 2019.

[19] W. Haensch, T. Gokmen, and R. Puri, "The Next Generation of Deep Learning Hardware: Analog Computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 108–122, 2019.

[20] A. Amirsoleimani, F. Alibart, V. Yon, J. Xu, M. R. Pazhouhandeh, S. Ecoffey, Y. Beilliard, R. Genov, and D. Drouin, "In-Memory Vector-Matrix Multiplication in Monolithic Complementary Metal–Oxide–Semiconductor-Memristor Integrated Circuits: Design Choices, Challenges, and Perspectives," *Advanced Intelligent Systems*, vol. 2, no. 11, p. 2000115, 2020.

[21] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: Design considerations," *Frontiers in neuroscience*, vol. 10, p. 333, 2016.

[22] T. Gokmen, M. Onen, and W. Haensch, "Training Deep Convolutional Neural Networks with Resistive Cross-Point Devices," *Frontiers in Neuroscience*, vol. 11, p. 538, 2017.

[23] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[24] F. Zahoor, T. Z. Azni Zulkifli, and F. A. Khanday, "Resistive random access memory (RRAM): an overview of materials, switching mechanism, performance, multilevel cell (MLC) storage, modeling, and applications," *Nanoscale research letters*, vol. 15,

pp. 1–26, 2020.

[25] O. Krestinskaya, A. Irmanova, and A. P. James, "Memristive non-idealities: Is there any practical implications for designing neural network chips?," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2019.

[26] H.-K. Liu, D. Chen, H. Jin, X.-F. Liao, B. He, K. Hu, and Y. Zhang, "A Survey of Non-Volatile Main Memory Technologies: State-of-the-Arts, Practices, and Future Directions," *Journal of Computer Science and Technology*, vol. 36, no. 1, pp. 4–32, 2021.

[27] C. You and H. Kim, "Effect of finite tunneling magneto-resistance for the switching dynamics in the spin transfer torque magnetic tunneling junctions," in *2017 IEEE International Magnetics Conference (INTERMAG)*, pp. 1–2, 2017.

[28] P. Chi, S. Li, Yuanqing Cheng, Yu Lu, S. H. Kang, and Y. Xie, "Architecture design with STT-RAM: Opportunities and challenges," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 109–114, 2016.

[29] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 256–267, IEEE, 2013.

[30] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[31] S. Yu, X. Guan, and H.-S. P. Wong, "On the stochastic nature of resistive switching in metal oxide RRAM: Physical modeling, Monte Carlo simulation, and experimental characterization," in *2011 International Electron Devices Meeting*, pp. 17–3, IEEE, 2011.

[32] S. Yu, Y. Wu, and H.-S. P. Wong, "Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory," *Applied Physics Letters*, vol. 98, no. 10, p. 103514, 2011.

[33] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[34] M. Zhang, S. Long, G. Wang, Y. Li, X. Xu, H. Liu, R. Liu, M. Wang, C. Li, P. Sun, *et al.*, "An overview of the switching parameter variation of RRAM," *Chinese science bulletin*, vol. 59, no. 36, pp. 5324–5337, 2014.

[35] C. Zambelli, A. Grossi, P. Olivo, C. Walczyk, and C. Wenger, "RRAM Reliability/Performance Characterization through Array Architectures Investigations," in *2015 IEEE Computer Society Annual Symposium on VLSI*, pp. 327–332, 2015.

[36] W. Chen, W. Lu, B. Long, Y. Li, D. Gilmer, G. Bersuker, S. Bhunia, and R. Jha,

"Switching characteristics of W/Zr/HfO2/TiN ReRAM devices for multi-level cell non-volatile memory applications," *Semiconductor Science and Technology*, vol. 30, no. 7, p. 075002, 2015.

[37] Z. Shen, C. Zhao, Y. Qi, W. Xu, Y. Liu, I. Z. Mitrovic, L. Yang, and C. Zhao, "Advances of RRAM Devices: Resistive Switching Mechanisms, Materials and Bionic Synaptic Application," *Nanomaterials*, vol. 10, no. 8, p. 1437, 2020.

[38] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: Variation-aware training for memristor x-bar," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[39] P.-Y. Chen, B. Lin, I.-T. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J.-s. Seo, Y. Cao, and S. Yu, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 194–199, IEEE, 2015.

[40] H. A. D. Nguyen, J. Yu, M. A. Lebdeh, M. Taouil, S. Hamdioui, and F. Catthoor, "A classification of memory-centric computing," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 2, pp. 1–26, 2020.

[41] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs, L. Xie, N. Wald, S. Joshi, H. M. Elsayed, H. Corporaal, and K. Bertels, "Memristor for computing: Myth or reality?," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 722–731, IEEE, 2017.

[42] Q. Xia and J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature Materials*, vol. 18, pp. 309–323, 2019.

[43] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[44] L. Chua, "If it's pinched it's a memristor," *Semiconductor Science and Technology*, vol. 29, p. 104001, sep 2014.

[45] J. Yu, H. A. Du Nguyen, L. Xie, M. Taouil, and S. Hamdioui, "Memristive devices for computation-in-memory," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1646–1651, IEEE, 2018.

[46] A. Prakash, J.-S. Park, J. Song, S.-J. Lim, J.-H. Park, J. Woo, E. Cha, and H. Hwang, "Multi-state resistance switching and variability analysis of HfOx based RRAM for ultra-high density memory applications," *2015 International Symposium on Next-Generation Electronics (ISNE)*, pp. 1–2, 2015.

[47] Y. Huang, Z. Shen, Y. Wu, X. Wang, S. Zhang, X. Shi, and H. Zeng, "Amorphous ZnO based resistive random access memory," *RSC Advances*, vol. 6, pp. 17867–17872, 2016.

[48] S. K. Gonugondla, A. D. Patil, and N. R. Shanbhag, "SWIPE: Enhancing Robustness of ReRAM Crossbars for In-memory Computing," *2020 IEEE/ACM International*

Conference On Computer Aided Design (ICCAD), pp. 1–9, 2020.

[49] D. Kumar, R. Aluguri, U. Chand, and T.-Y. Tseng, "Metal oxide resistive switching memory: materials, properties and switching mechanisms," *Ceramics International*, vol. 43, pp. S547–S556, 2017.

[50] W. Shim, Y. Luo, J.-S. Seo, and S. Yu, "Investigation of read disturb and bipolar read scheme on multilevel RRAM-based deep learning inference engine," *IEEE Transactions on Electron Devices*, vol. 67, no. 6, pp. 2318–2323, 2020.

[51] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176–183, 2013.

[52] Z. Chen, C. Schoeny, and L. Dolecek, "Coding assisted adaptive thresholding for sneak-path mitigation in resistive memories," in *2018 IEEE Information Theory Workshop (ITW)*, pp. 1–5, IEEE, 2018.

[53] G. Song, K. Cai, C. Sun, X. Zhong, and J. Cheng, "Near-Optimal Detection for Both Data and Sneak-Path Interference in Resistive Memories with Random Cell Selector Failures," *arXiv preprint arXiv:2101.09680*, 2021.

[54] Y. Ben-Hur and Y. Cassuto, "Detection and coding schemes for sneak-path interference in resistive memory arrays," *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 3821–3833, 2019.

[55] Z. He, J. Lin, R. Ewetz, J. Yuan, and D. Fan, "Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019.

[56] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," *SIGARCH Comput. Archit. News*, vol. 44, p. 27–39, June 2016.

[57] P.-Y. Chen, X. Peng, and S. 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.

[58] S. Gupta, M. Imani, J. Sim, A. Huang, F. Wu, M. H. Najafi, and T. Simunic, "SCRIMP: A General Stochastic Computing Architecture using ReRAM in-Memory Processing," *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1598–1601, 2020.

[59] S. Joshi, C. Kim, S. Ha, and G. Cauwenberghs, "From algorithms to devices: Enabling machine learning through ultra-low-power vlsi mixed-signal array processing," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–9, 2017.

[60] Y. Kim, S. Kim, C.-C. Yeh, V. Narayanan, and J. Choi, "Hardware and Software Co-optimization for the Initialization Failure of the ReRAM-based Cross-bar Array," *ACM*

Journal on Emerging Technologies in Computing Systems (JETC), vol. 16, no. 4, pp. 1–19, 2020.

[61] C. Lammie, W. Xiang, B. Linares-Barranco, and M. R. Azghadi, "MemTorch: An Open-source Simulation Framework for Memristive Deep Learning Systems," 2020.

[62] S. Huang, A. Ankit, P. Silveira, R. Antunes, S. R. Chalamalasetti, I. El Hajj, D. E. Kim, G. Aguiar, P. Bruel, S. Serebryakov, *et al.*, "Mixed Precision Quantization for ReRAM-based DNN Inference Accelerators," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 372–377, 2021.

[63] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[64] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[65] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," 2018.

[66] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," 2017.

[67] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.