LUND UNIVERSITY

LTH FACULTY OF ENGINEERING

Evaluation of Ferroelectric Tunnel Junction memristor for in-memory computation in real world use cases

Alec Guerin & Christos Papadopoulos

Master of Embedded Electronics Engineering
Project duration: 4 months

ERICSSON

Supervised by Mattias Borg
Co-Supervised by Saeed Bastani

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
Host company: Ericsson
May 2023

Abstract

Machine learning algorithms are experiencing unprecedented attention, but their inherent computational complexity leads to high energy consumption. However, a paradigm shift in computing methods has the potential to address the issue. This shift could be a move towards analog in-memory computing, a method which uses Ohm's and Kirchhoff's Laws, and carries out the processing directly where data resides. This approach is being propelled by the development of memristors, versatile memory devices that are programmable and energy efficient.

This thesis explores the capabilities of a newly engineered memristor device. This device, based on Ferroelectric Tunnel Junctions (FTJ), was developed by Lund University and presents promising technology for analog in-memory computing. In this thesis, the creation of a mathematical model took place within a simulated setting. This provided the foundation for a sensitivity analysis of chosen neural network algorithms operating on hardware featuring FTJ devices. A variety of techniques were deployed to mitigate the hardware imperfections, such as hardware-aware training, which enhanced the resilience of the algorithms.

The outcomes from this investigative approach are promising, particularly regarding the inference processes in neural networks. Our research demonstrated the effectiveness of all applied mitigation techniques. The standout discovery was the robustness of the Transformer algorithm, compared to convolutional one, which proved capable of withstanding hardware imperfections while producing results on par with those of the digital model.

Popular Science Summary

Machine learning algorithms have become an integral part of our daily lives, powering applications and systems across various industries. However, their widespread use comes with a significant drawback - high power consumption. As the demand for machine learning continues to grow, there is an urgent need for innovative computing approaches that can tackle this power challenge. Enter Analog In-Memory Computing (AIMC), a promising solution that could revolutionize the way we process information. When combined with a type of memory cell called a Ferroelectric Tunnel Junction (FTJ) memristor, AIMC can perform the calculations needed for machine learning with much less power.

In this study, we evaluated the effectiveness of FTJ memristors, engineered at NanoLund, in enabling AIMC. Our findings were encouraging. Interestingly, FTJ memristors demonstrated impressive performance when interfaced with the same kind of transformer neural network that powers ChatGPT, a state-of-the-art algorithm. But our work did not stop there. We also explored the potential of AIMC for training neural networks and designed several techniques to enhance the precision of our networks. Finding the proper way to map your application on the AIMC hardware is quite the challenge. Furthermore, we analyzed the sensitivity of the algortihms and proposed approaches that could make them functional. Through this comprehensive understanding, we suggest the implementation of AIMC for transformer models and pinpoint the key attributes for a successful design, known as mitigation techniques.

The magic of AIMC lies in its simplicity. It's based on Ohm's and Kirchhoff laws for multiplication and addition. These combined operations, known as multiplication accumulation (MAC), are extensively used for matrix operations required for machine learning algorithms. While the digital domain needs extensive hardware and power consumption to compute the operation, in the analog domain, simple resistors can do the trick.

So, why should you care? Because this revolution in machine learning could transform everything from artificial intelligence to data processing and beyond. It could make our technologies smarter, our data more meaningful, and our lives better.

Acknowledgements

We wish to express our deep gratitude to our supervisors, Professor Mattias Borg from Lund university and Dr. Saeed Bastani from Ericsson Research, for their invaluable support, guidance and for providing us with the chance to undertake this thesis. Their insights and assistance have been key in our research.

We would also like to thank Ericsson Lund for supporting this thesis topic and providing us with the necessary resources to undertake this project. Furthermore, we would like to express our sincere gratitude to all our professors in Lund University, specifically in the Faculty of Engineering (LTH), for providing us with the knowledge and skills necessary to conduct this thesis.

Credit should also be given to the creators of AIHWKIT and YOLOv8, state-of-the-art tools in the field, for making them open-source, freely available and continuously maintaining them for the community. Their support through the forums of these tools was also significant.

Lastly, we wish to convey our deepest appreciation to our families, whose unwavering support has been invaluable. We also extend thanks to our friends, who provided the much-needed relief during stressful times, and our partners, for their daily encouragement and backing. Moreover, we are thankful to each other for the collaboration, shared laughs, and the enjoyable journey we embarked on together throughout this entire process.

Abbreviations

- AIMC : Analog In-Memory Computing
- NN : Neural Network
- CNN : Convolutional Neural Network
- DNN : Deep Neural Network
- SGD : Stochastic Gradient Descent
- FTJ : Ferroelectric Tunnel Junction
- PCM : Phase Change Memory
- RPU : Resistive Processing Unit
- VMM : Vector Matrix Multiplication
- FP32 : Floating Point 32 bits
- MAC : Multiply Accumulate
- GPU : Graphical Processing Unit
- tGDP : Tile Gradient Descent Programming
- ADC : Analog to Digital Converter
- DAC : Digital to Analog Converter
- TPU : Tensor Processing Unit
- HWA : Hardware-Aware
- API : Application Programming Interface
- DFL : Distribution Focal Loss
- RGB : Red-Green-Blue
- ENOB: Effective Number Of Bit

Contents

Chapter 1

Introduction

Charles Babbage, often referred to as the "father of the computer", designed the first mechanical computer in the 19th century [1]. While his invention was not realized, it nonetheless laid the groundwork for modern computing. During 20th century, the thriving demand for scientific computing led to the development of analog computers. These early devices, typically mechanical or electrical, were constructed around specific problems, thus limiting their flexibility and accuracy compared to contemporary digital computers.

The trajectory of computing technology continued to evolve, transitioning from vacuum tube-based machines to transistor-based ones following their invention at Bell Labs [2]. The field subsequently experienced an exponential surge in technological advancements. Gordon Moore, co-founder of Intel, famously predicted that the number of transistors on an integrated circuit would double annually, a trend that persisted for several decades, now known as Moore's Law. However, engineers hit a physical limitation in scaling down transistor size due to the increasing prominence of quantum mechanical effects and the challenges posed by current leakage, a significant source of power consumption. Hence, a requirement for a new computing paradigm to propel the advancement of computing was needed.

A notable paradigm shift in current technology involves reverting to a strategy employed in the past — designing computers tailored to solve specific problems. This approach typically involves the use of a general-purpose processor in conjunction with a specialized accelerator, aimed at enhancing the speed of a particular processing element. Such architectural configurations are now predominantly employed in machine learning (ML) algorithms, notably in neural networks (NN). However, a significant challenge that must be addressed in this context is the issue of data transfer, which presents a substantial energy and latency bottleneck in these systems, especially for data-intensive operations like neural networks. A neural network running on an Nvidia 3090RTX graphics processing unit (GPU) can consume about 370 Watts, while the human brain, performing a multitude of tasks simultaneously, uses only around 20 Watts [3].

Taking inspiration from the human brain, in-memory computing is an approach that could resemble it. Instead of transferring the data to a processing unit, known as Von Neumann architecture, the computation could be performed where the data are, limiting the transfer bottleneck. Meanwhile, looking back again in the past, analog computing should be reconsidered. In 1971, Leon Chua introduced the concept of a memristor, a type of passive circuit element that maintains a resistance dependent on the history of the current that has passed through it [4]. This concept remained theoretical until 2008, when the first such device was realized at HP Labs [5]. This type of device is a promising candidate for the technique of analog in-memory computing. Leveraging non-volatile resistive memory devices in an array format, one can execute multiplication and accumulation (MAC) operations using Ohm's and Kirchhoff's laws fast and at a low power cost. In this way, the need for data movement is substantially reduced, hence boosting processing speed and energy-efficiency.

While the potential of this method is intriguing, it also comes with a set of challenges. Present-day memristors often struggle with issues like noise disruption, limited resistive states, unstable state programming, and variations in resistance over time. Moreover, the circuitry that supports these memristors could negatively impact the precision of the results. The objective of this study is to address these complications and determine if a particular device, the Ferroelectric Tunnel Junction (FTJ), can perform satisfactorily when employed in neural network algorithms. This research will also assess how different algorithms respond to the Analog In-Memory Computing (AIMC) approach to pinpoint the most suitable ones.

The decision to use FTJ is based on several advantages: its high resistance implies energy efficiency; it has adaptability in terms of size; it can sustain its resistive state for a long time; and it complies with prevalent CMOS technology, a common method for creating integrated circuits.

The focus of this thesis lies in the application of an FTJ-based memristor for AIMC in machine learning. With its roots at Nanolund, the performance of the FTJ device is evaluated within various network structures, from basic to advanced, in a simulation environment established by IBM [6].

This thesis is separated in four parts. It starts by the scientific background which aims to provide a deeper understanding of the key concepts and background relevant to the research conducted in this work. Next the methods employed during this work are described. This chapter aims to provide a comprehensive overview of the key aspects involved in the research conducted on the FTJ memristors. Following the methods, the results will be presented to provide an evaluation of the potential and limitations the FTJs device for AIMC. Finally, the limitations, possible improvements, and other important considerations for AIMC will be discussed in the discussion part.

Simulated results suggest that despite the imperfections associated with analog computing, certain neural network algorithms can achieve ISO-standard accuracy using the FTJ device. Hence, additional research and practical implementation are recommended.

Our main contributions to Analog In-Memory Computing (AIMC) research are:

1. Creation and verification of FTJ models in simulation enviroment.
2. Evaluation of the FTJ technology's potential for neural network analog in-memory computing, both for inference and training.
3. Identification of the need for a precise weight programming method and proposal of techniques.
4. Investigation of synaptic mixed-signal slicing techniques.
5. Analysis of the sensitivity of the neural networks on the choice of AIMC hardware

1.1 Stakeholders

This thesis project was proposed and hosted by the department of Device Platform Research (DPR) at Ericsson, Lund, Sweden. The company proposed the research topic, which focused on evaluating FTJs memristors for state-of-the-art machine learning algorithms. Ericsson also provided essential resources and support for the successful completion of this project. The FTJ models and data used in in the research were provided by NanoLund. Table 1.1 presents the stakeholders involved in this project and outlines the benefits associated with their involvement.

Table 1.1: Stakeholders involved in this thesis and their benefits

Stakeholders	Benefits
Ericsson Lund	1. Insight of the usage of the FTJs memristor, potential and limitation. 2. Access to the projects we developed to explore the usage of FTJ.
NanoLund	1. Evaluation of the technology they developed. 2. Proposals for future work.

Chapter 2

Scientific Background

In this chapter, the various subjects addressed in this work will be discussed. We will begin by presenting the different types of neural networks used. Next, we will introduce the memristor device and more precisely the FTJ. Following the introduction of memristors, we will explore the concept of AIMC and how to address some of its challenges.

By covering the topics of neural network types, memristor devices, and AIMC, this chapter aims to provide a comprehensive understanding of the key concepts and background relevant to the research conducted in this work.

2.1 Neural Networks

Recent breakthroughs in machine learning have resulted in the creation of sophisticated learning systems that heavily rely on neural networks. These systems, often called simulated or artificial neural networks, have gained significant recognition for their capacity to analyze vast amounts of data and, in some cases, even produce images and text.

Artificial neural networks consist of interconnected layers of nodes, including an input layer, one or more hidden layers, and an output layer (2.1). These layers work together, with individual nodes representing artificial neurons. Each neuron is connected to other neurons, where each connection has its own weight learned during the training of the neural network. Also, each neuron includes an activation function, often characterized by function type (such as Rectified Linear Unit–ReLU) and a threshold. If a node's output surpasses the specified threshold, it becomes activated and passes data to the next layer. Conversely, if the output falls below the threshold, data transmission is restrained.

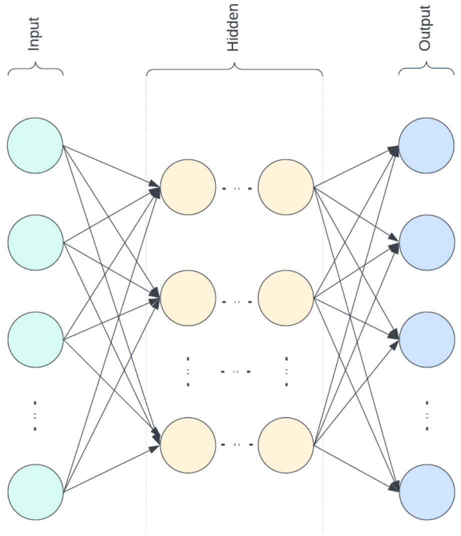


Figure 2.1: Artificial Neural Network graph

The mathematical model of a neuron is similar to linear regression, expressed as follows:

$$y_i = \sum_{j}^{N} w_{i,j} x_j + b_i \ , \tag{2.1}$$

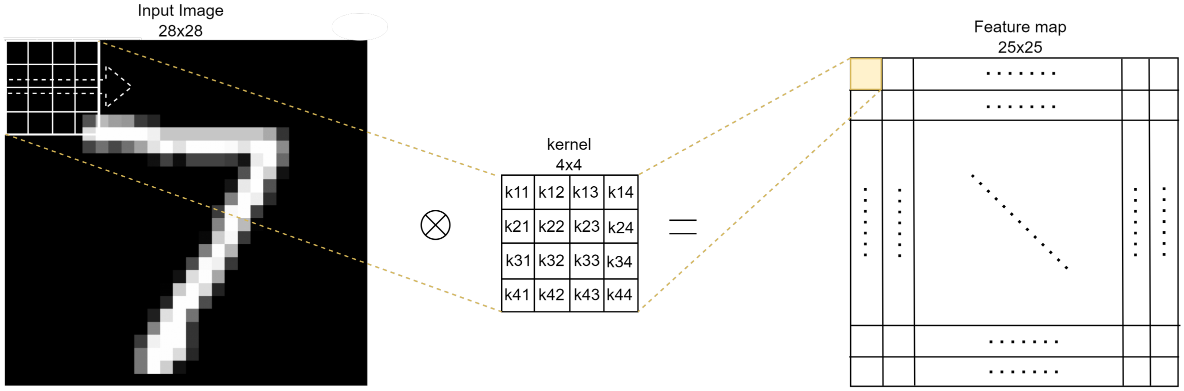
where y_i and x_j are the neuron's activation output and inputs, $w_{i,j}$ are the weights of each connection to i^{th} neuron and b_i a bias of the neuron. Afterward, the output is passed through an activation function, which determines the output, often through a non-linear function (or non-linearity, for short). If that output exceeds a given threshold, it "fires" (or activates) the neuron, passing data to the next layer in the network. This results in the output of one neuron becoming the input of the next neuron. When all the neurons from a layer have connections with all the outputs from the previous layer, then it is called a fully connected layer. This process of passing data from one layer to the next layer defines this neural network as a feedforward network.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of feedforward artificial neural network that is primarily used for classifying images [7]. They are designed to automatically and adaptively learn features from input data. These networks are particularly useful for tasks like image and video recognition, as well as other applications that rely heavily on the extraction of features from raw input data. Their first appearance happened in 1980 when Kunihiko Fukushima introduced the "neocognitron" [8], but were popularized by AlexNet in 2012 [9].

CNNs are principally built with the combination of convolutional layers, pooling layers and fully connected layers [10]. The convolutional layers consist of kernels that are applied to the output of the previous layer in a convolution operation. In convolution, as shown in 2.2, each kernel (also known as a filter) slides over the input data (like a patch of an image), performing a dot product operation between the kernel's weights and the input data in its current position

[7]. This operation results in a feature map, which is a matrix that represents which features the kernel has detected in the input data.

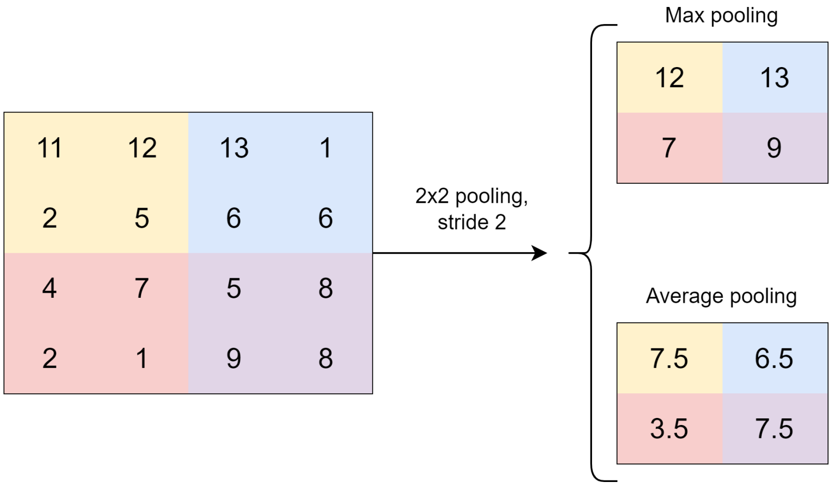


k11	k12	k13	k14
k21	k22	k23	k24
k31	k32	k33	k34
k41	k42	k43	k44

Figure 2.2: Convolution of a 4x4 kernel on an 28x28 image with 0 padding and stride of 1.

The size of the kernels, the stride (the step size when moving the kernel), and the padding (adding extra zeros around the input data) are parameters that can be adjusted to alter the dimensionality and other properties of the feature maps. Each kernel is specialized in detecting a specific feature or pattern, like edges, textures, or colors, in the input data.

Next in the architecture of a CNN are the pooling layers. Pooling layers are used to reduce the dimensionality of the feature maps while preserving the most important information. The two most common types of pooling are max pooling and average pooling. Max pooling takes the maximum value from a section of the feature map, while average pooling takes the average of the values in that section. Figure 2.3 shows a simple example of 2x2 pooling with stride 2. This dimensionality reduction helps to reduce computational requirements and also helps to control overfitting, which occurs when a model learns the training data too well and performs poorly on unseen data.



11	12	13	1
2	5	6	6
4	7	5	8
2	1	9	8

Max pooling

12	13
7	9

Average pooling

7.5	6.5
3.5	7.5

Figure 2.3: Pooling of 2x2 and stride of 2.

Finally, we have fully connected layers. Fully connected layers are similar to those in traditional multilayer perceptron neural networks[11]. Each neuron in a fully connected layer is connected to every neuron in the previous layer. The fully connected layers usually come towards the end of a CNN, and they aim to learn the non-linear combinations of the high-level features represented by the output of the previous layer.

The final fully connected layer often has the same number of neurons as there are classes for the classification task. The output of this layer is then passed through a softmax function, which converts the outputs into probability scores for each class, where the class with the highest score is selected as the output prediction of the model.

2.1.2 Transformers

In 2017, a group of researchers, led by a scientist named Vaswani, introduced a new way for computers to understand and generate human language. They called it the 'Transformer' model, as described in their paper "Attention is All You Need" [12]. Unlike other models that were popular at the time, which processed language word-by-word in order (like we read a book), or looked for patterns in blocks of text (like searching for themes in a paragraph), the Transformer model could focus on different parts of a sentence at the same time, to understand the context better. This was important, particularly for translation tasks, and it also helped in other areas where computers interact with human language, images, and audio.

The Transformer model has two main parts: the 'encoder' and the 'decoder'. These parts are built with a stack of identical layers, just like a multi-story building has many identical floors. The encoder takes the input (like a sentence in English), and the decoder produces the output (like the translated sentence in French). In addition, the input data are preprocessed by the input embedding and positional encoding system. The input embedding transforms the words into vectors in order for the computer to be able to understand them. While the positional encoding provides information of where in the sentence the word is located.

The main job of the encoder is to create a mathematical representation, a tensor, of the input that contains language understanding and context meaning. This is done by the combination of a multi-head attention module and feed-forward network. The details of this are quite intricate but mainly the attention module tries to calculate the importance of one word with itself and others. After encoding all the needed information, it feeds it to the decoder which uses similar modules and predicts the next possible word, in the case of sentence translation, as output. The mathematical operations done in multi-head attention module and feed-forward network are both plain matrix multiplications. This means that this modules can be efficiently processed by the later mentioned analog in memory technique.

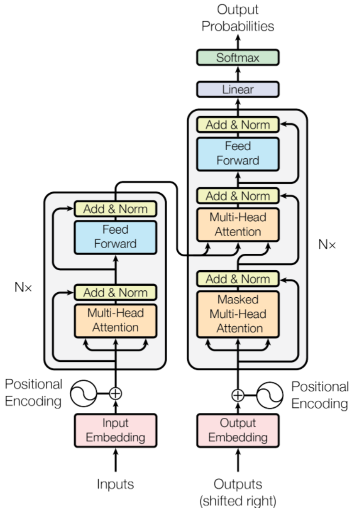


Figure 2.4: The Transformer - model architecture. Source [12]

In short, the Transformer model brought a new way for machines to understand and generate human language by focusing on different parts of a sentence at the same time. It has been a foundation for many recent advancements in areas like language translation, content generation, and even image recognition. Despite its advantages, researchers are still working on making it even better, particularly in making it more efficient to run on computers.

2.2 Memristors

Analog resistive memory devices, or memristors, are nanoelectronic devices whose electrical resistance depends on the history of voltage applied to them [13].

$$V(t) = M(q(t))I(t), \tag{2.2}$$

where $V(t)$ is the voltage at time t, M(q(t)) is the memristance (a function of the charge passed through the device) and I the current.

They have been proposed as a potential building block for non-volatile memory devices, which are capable of retaining data when the power is turned off. Their unique characteristics make them attractive for acceleration of various algorithms such as neural networks and matrix algebra for applications in the domains of machine vision, media and signal processing, to name a few. Currently, various devices have been proposed as memristor candidates and the research is ongoing.

2.2.1 Ferroelectric Tunnel Junction

In this study, we focus on a specific type of memristor device called Ferroelectric Tunnel Junction (FTJ). FTJ devices offer several advantages, including low currents, long retention, multi-level operation and ultra-low switching energy. Experimental FTJ devices have been fabricated at LTH [14], and their characteristics have been modeled. However, the performance of FTJs for analog in-memory computing on real machine learning use cases remains unexplored. The availability of a model for these devices presents an opportunity to investigate their potential in this domain. In the scope of this thesis, the ferroelectric material of choice is a $Hf_xZr_{1-x}O_2$. The material is squeezed usually between two metal layers and creates a barrier for the electrons who travel between them. Taking advantage of the ferroelectricity of the material, the effective resistance of the device can be switched. The mechanisms that play a role in this device are explored next.

Polarization and Ferroelectricity

To comprehend the behavior of FTJ devices, it is essential to delve into the concepts of polarization and ferroelectricity. Electrical polarization refers to the phenomenon that occurs in a dielectric material when an external electric field is applied. The field influences the movement of electrons or charged atoms within the material, causing them to align in the direction of the field. Consequently, a dipole is formed, resulting in opposite charges on each side of the material. This process creates an intrinsic electric field in the material, known as polarization charge (Figure 2.5).

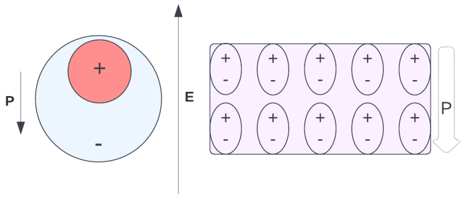


Figure 2.5: Polarization of an atom and a dielectric material

The electric polarization charge of a material can be expressed as:

$$P = \epsilon_0 \chi E \tag{2.3}$$

where ϵ_0 represents the vacuum permittivity, χ denotes the electric susceptibility of the dielectric material, and E signifies the applied electric field. Furthermore, the electric susceptibility is related to the relative electric permittivity (ϵ_r) of a material according to:

$$\epsilon_r = 1 + \chi \tag{2.4}$$

Hence, any material with a relative permittivity greater than one can be polarized.

Ferroelectricity is a unique characteristic exhibited by certain materials, enabling them to maintain a stable polarization even in the absence of an external electric field. This property arises from the inherent asymmetry of the crystal structure in ferroelectric materials. The arrangement of atoms or ions within these materials results in two or more stable equilibrium positions, leading to the presence of spontaneous polarization. Consequently, the polarization can exist independently of any external electric field. However, when an external electric field is applied to a ferroelectric material, it can induce a reorientation or switching of the polarization direction.

Ferroelectrics were first discovered more than a century ago [15], marking a significant milestone in the field. However, it wasn't until 2011 [16] that the revelation of ferroelectric properties in hafnium oxide under specific conditions breathed new life into research focused on these particular devices. This groundbreaking discovery paved the way for further investigations and advancements in the field of ferroelectric tunnel junctions.

Electron Transport in FTJ

There are three main desired phenomena of electorn transport across the FTJ device. All of them are based on the potential energy barrier that the oxide introduces between the metals. Due to the polarization of the ferroelectric this barrier can be changed and as a result introduce different resistance for the current that goes through the device. By understanding the mechanisms for the transport, it becomes clear how ferroelectricity affects them.

The first mechanism is thermionic emission. In this case, the electron finds enough energy to overcome the potential barrier and travel through the oxide 2.6a. This mechanism is heavily influenced by the temperature of the system and the height of the potential barrier [23].

The second and third mechanisms are Direct tunneling 2.6b and Fowler-Nordheim Tunneling 2.6c, which are based on the quantum mechanical phenomenon of tunneling. In these cases, the electron does not have enough energy to overcome the barrier, but due to the wave-like nature of particles, it can still pass through the barrier. The probability of this happening is determined by the thickness and height of the barrier which can be influenced by the polarization of the ferroelectric material.

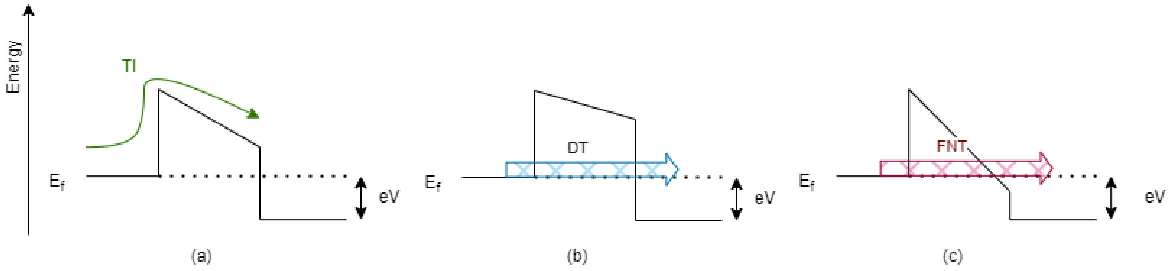


Figure 2.6: Transport Mechanisms in FTJ devices: a) Thermionic Emission, b) Direct Tunneling, c) Fowler-Nordheim Tunneling.

The paper by Pantel and Alexe (2010)[17] provides an analysis of the electroresistance effects in ferroelectric tunnel barriers, which is an important characteristic of FTJ devices. They found that the resistance of the junction can be changed by the polarization state of the ferroelectric barrier, which is consistent with the mechanisms described above [17].

2.3 AIMC

The existing methodology for analog in-memory computing has centered around employing Resistive Processing Unit (RPU) crossbar arrays, also known as tiles, for the calculations of a neural network layer. This architecture uses the principles of Ohm's law and Kirchhoff's current law to perform vector matrix multiplication (VMM) calculations. As illustrated in Figure 2.7, the matrix weights are mapped onto the resistances of memristors, while the inputs are typically encoded as voltages using digital-to-analog converters (DAC), based on the designer's choice. The outcome is an accumulated current, which can later be converted to a digital representation using analog-to-digital converters (ADC).

It is important to mention that both the input and weight values are standardized to fall within the range of -1 to 1. Since conductance is always positive, creating negative weights necessitates the use of an additional parallel tile. This second tile receives only inverse input pulses, thus achieving the desired effect.

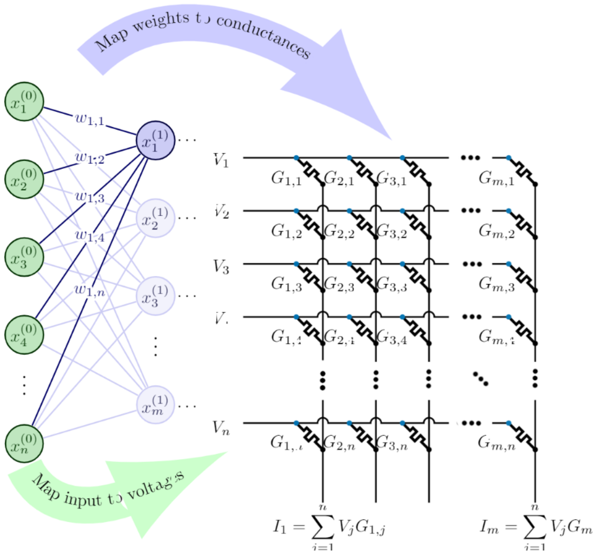

Map weights to conductances
$x_1^{(0)}$
$x_2^{(0)}$
$x_3^{(0)}$
$x_4^{(0)}$
$x_n^{(0)}$
$w_{1,1}$
$w_{1,2}$
$w_{1,3}$
$w_{1,4}$
$w_{1,n}$
$x_1^{(1)}$
$x_2^{(1)}$
$x_3^{(1)}$
$x_m^{(1)}$
V_1
V_2
V_3
V_4
V_n
$G_{1,1}$ $G_{2,1}$ $G_{3,1}$ $G_{m,1}$
$G_{1,2}$ $G_{2,2}$ $G_{3,2}$ $G_{m,2}$
$G_{1,3}$ $G_{2,3}$ $G_{3,3}$ $G_{m,3}$
$G_{1,4}$ $G_{2,4}$ $G_{3,4}$ $G_{m,4}$
$G_{1,n}$ $G_{2,n}$ $G_{3,n}$ $G_{m,n}$
$I_1 = \sum_{i=1}^{n} V_j G_{1,j}$
$I_m = \sum_{i=1}^{n} V_j G_m$
Map input to voltages

Figure 2.7: Feedforward Neural Network mapped to RPU crossbar array. Figure source: [18]

2.3.1 Power consumption

Reducing power consumption in today's society has become an important part of the fight against global warming. According to Saul J, Bass D. Bloomberg article [19], Google spent up to 10 to 15% of its total electricity consumption on AI, which could represent 2.3 TWh in 2021. For reference, Sweden consumed 136.7 TWh in 2017. AI is a rapidly growing field and is becoming increasingly widespread. The energy used for training models is significant, often requiring multiple GPUs/accelerators to run for several days. The power consumption during inference is also increasing rapidly with the growing use of services like Chat GPT. Reducing power consumption also lowers costs, increases scalability, and enables embedding AI in smaller consumer devices and IoT (Internet of Things).

The most common architecture for computation is based on separated memory and computing units linked by buses. The data moves from memory to computing units where they are processed and then moved back to the memory. This back-and-forth movement with the memory has become the main speed limitation, as so called the von Neumann bottleneck. It not only limits computation speed but also represents a significant portion of power consumption.

The training and inference of Deep Neural Networks (DNNs) heavily rely on matrix algebra, including VMM operations, which requires a significant amount of energy when computed in the digital domain using FP32. In classical architectures, both operands of an operation need to be fetched from memory to the processing unit and the result written back into memory. While GPUs have accelerated these operations by parallelization, they have not solved the data transfer and power consumption issues. This high-power consumption limits the possibility of embedding advanced DNNs, for example in mobile applications.

To address this issue, AIMC has been proposed to reduce these two main sources of power consumption. AIMC performs MAC operations in the analog domain using Ohm's and Kirchhoff's laws, which are fast and power-efficient. The second power reduction is achieved by using memristors to store the model's weights, allowing only the input to change during inference and update on chip for training. Although going to the analog domain may result in drastic reduction of precision but DNNs have shown resilience to it.

In current AIMC implementations, the peripheral circuitry often limits the advantages obtained by memristors by consuming a significant share of energy and area compared to the resistive crossbar array using low voltage and high resistance. To communicate with the digital world, to which the accelerators are connected, data needs to be converted from one domain to the other. This requires each row of the crossbar array to have access to a DAC, and each column needs access to an ADC. Strategies for input and weight management can help reducing the precision and the amount of peripheral circuitry. The ADC is the most sensitive part since it reads the accumulated value across a whole RPU column, while the DAC impacts only one RPU current seen by the ADC. Studies have shown that using 6-bit DAC and 8-bit ADC yields acceptable results. For example, ADC consumes 49% of the total chip power in ISAAC implementation [20] and 41% in the memristive Boltzmann machine [21].

There are two commonly used ADC architectures for AIMC: the ramp ADC and the successive approximation register (SAR) ADC. The ramp ADC compares the analog signal to a linearly increasing reference voltage generated by a ramp generator. It is suitable for highly parallel comparisons in the crossbar, with the entire array sharing a single ramp generator. However, its speed and its energy consumption are dominated by the multiple comparators needed. The latency scales as $O(2^B)$, where B is the bit count, but does not increase with the number of columns. The ADC consumption for a VMM scales as $O(N_c 2^B)$, where N_c is the column count, assuming constant power consumption during the ramp time [22].

On the other hand, the SAR ADC uses a binary search algorithm with a B-bit DAC to find the correct digital output. Due to its large area, a single SAR ADC is typically shared by multiple crossbar columns via time multiplexing. The SAR ADC's latency scales with the resolution and the number of columns, $O(BN_c)$, while its power consumption and area scale exponentially with the number of bits, $O(BN_c 2^B)$ [22].

The choice between the ramp ADC and the SAR ADC depends on the desired resolution and the number of columns. Other ADC types like flash ADC, delta-sigma ADC, or current control oscillator ADC can be considered for lower resolution or slower conversion.

The speed of the memristor crossbar array is also limited by the RC constant of the circuit, which depends on the technology used and its position in the chip. According to Gokmen and Vlasov [23], if the memristors can be implemented between metal layers, the propagation delay can be in the order of 100 ns. This allows the use of faster ADCs and multiplexing their inputs across multiple columns.

Table 2.1: ADC survey for 8 bit in order of 1GS/s

Type	GS/s	Techno [nm]	bit	ENOB	Power [mW]	Area [mm^2]	Cite
2 step SAR	1.2	65	8	6.97	5	0.013	[24]
SAR	1.2	32	8	6.24	3.06	0.0015	[25]
TB-sub	1	65	8	7.08	2.3	0.007	[26]
LU-SAR + Flash	0.9	28	8	7.36	1.88	0.0056	[27]
2 step SAR	1.1	40	8	7.18	4	0.00165	[28]

During inference, the power consumption of the crossbar array is mainly governed by the memristor heat dissipation. To estimate values for FTJ memristors, we take a reading voltage of 300mV and a mean resistance of 100M Ω [14] for a crossbar array of 512 by 512 RPUs and a 100% usage, the power consumption dissipated in the memristors is 0.236 mW [23] (eq.2.5). This value is way lower compared to an ADC running at 1GS/s (2-5mW) while the size of the crossbar array is small.

$$P_{\text{crossbar\_array}} = N \cdot M \cdot \frac{V^2}{R} \cdot \alpha \tag{2.5}$$

The FTJ small reading voltage and high resistance are positive to lower the power consumption but the resistance can also reduce the achievable speed. The propagation delay is limited by the time constant $\tau = RC$, where the resistance 'R' is composed of the transmission line and the memristor resistances and the capacitance 'C' is the transmission line parasitic capacitance. The transmission line width can not be reduced too much to make sure the IR drop is not too high and to allow the higher current needed to program the memristors. Another limitation is the time to integrate the current in a capacitor to be read by the ADC. The nanoampere range of the output current can be challenging to read due to it's proximity with the noise level. Therefor, this parameter should be taken into account when scaling the crossbar array size.

2.3.2 Non-Idealities

The main factors that may affect the use of memristive hardware technologies are known as non-idealities. The non-idealities source from the peripheral circuitry but also from the memristors themselves, as shown in 2.8. Starting from DAC, the precision of it introduces a quantization error that can impact the accuracy of a neural network. The positive aspect of this error is that it is deterministic and a neural network can adapt to it quite well [29]. However, non-deterministic errors such as clock jitters and imprecise pulse formations can give rise to undesired results. The same is true for the ADC, which not only has the same issues but also has a bounded range within which it can convert a current to an output, after which it clips the result. Overall, it will be apparent that the effect of the ADC is of significance and proper converters are important for the operation of this technology.

Regarding the interconnections of the crossbar array itself, there are noise sources from temperature, strain, sneak paths and voltage (IR) drop effects which degrade the outcome of the VMM. One of the most important aspects in this is IR drop, which is the voltage drop that occurs along a word line due to the finite resistance of the memory cells along it. It is one of the main reasons that the dimensions of a tile are restricted.

The issue of sneak path currents occurs when voltage is applied to a row in the array but given the grid-like structure of the array, there exist alternative or "sneak" paths through other memristors that the current may take. These sneak paths can result in the actual current flowing through the target memristor deviating from the anticipated current, thus leading to errors during data reading or writing operations. Various mitigation strategies exist for this issue, such as integrating a diode or transistor with the memristor. However, some memristors exhibit a non-linear current-voltage (I-V) characteristic, which eliminates the need for the previously mentioned selector.

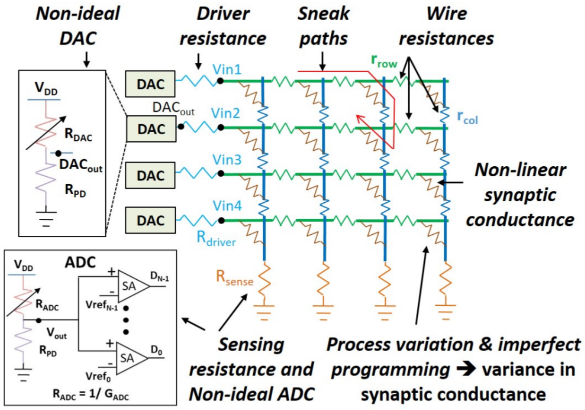


Figure 2.8: RPU crossbar array non-idiealities Figure source: [30]

In the memristor level, the non-idealites are various. Regarding the read process, the device can show variations due to 1/f noise, telegraph noise, temperature related effects, shot noise and conductance drift. Apart from this, the programming of the device creates errors towards the expected result. The device programming is completely dependent on its physics and the distinct resistive states of the device. Further more, non-optimized fabrication can lead to significant device to device variation. Due to the stochastic nature of quantum effects, the cycle to cycle response of a device to voltage pulses is also different. Overall, there are various factors that contribute to undesired performance and also should be taken into account during the verification of this technology.

2.3.3 Training with AIMC

Plain Stochastic Gradient Descent (SGD)

Training in AIMC is not a simple task. Gokmen and Vlasov [23] proposed a method for training in an RPU array using parallel pulsing scheme based on the Stochastic Gradient Descent (SGD) algorithm. Obviously, an array needs to have the necessary peripheral circuitry for backpropagation but also the memristors themselves have to be selected every time they have to be updated.

The update rule [31] is usually expressed as :

$$w_{ij} \leftarrow w_{ij} + \eta x_i \delta_j \tag{2.6}$$

where w_{ij} represents the weight value for the i^{th} row and the j^{th} column (for simplicity layer index is omitted) and x_i is the activity at the input neuron, δ_j is the error computed by the output neuron and η is the global learning rate.

In order to implement this update in an array, they proposed to simplify the multiplication to a stochastic computing technique, already known since 1967 [32]. It was shown that by using two stochastic bit streams the multiplication operation can be reduced in a simple AND operation between the bit streams. This technique is used in the case of the x_i and δ_j values. These arrays are translated by stochastic translators, a type of circuit that is not studied here, but which translates a floating point number into a stream of bits in which the probability of finding a 1 bit being the normalized floating point number. Then they are fed to the crossbar in parallel, one from the forward direction and the other from the backward direction. When they coincide on a memristor they potentiate or depress them by a certain amount Δw_{min}, as shown in Figure 2.9.

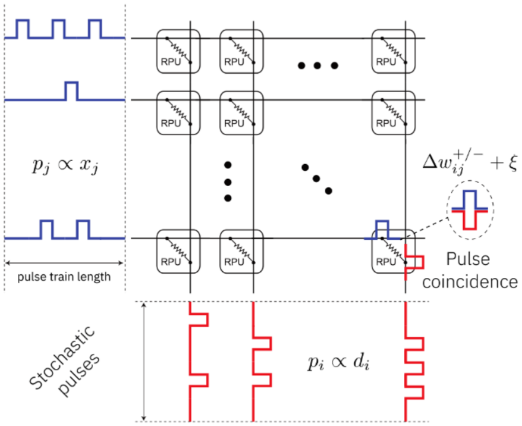


Figure 2.9: Fully Parallel Update using Stochastic Gradient Descent. Source[AIHWKIT]

Now the update rule is written as follows:

$$w_{ij} \leftarrow w_{ij} \pm \Delta w_{min} \sum_{n=1}^{BL} A_i^n \wedge B_j^n \tag{2.7}$$

BL is the length of the stochastic bit stream, Δw_{min} is the change in the weight value due to a single coincidence event, A_i^n and B_j^n are random variables that are characterized by a Bernoulli process, and the superscript n represents the bit position in the stream. The probabilities that A_i^n and B_j^n are equal to unity are given by Cx_i and $C\delta_j$, respectively, where C is a gain factor in the stochastic translator circuit and is calculated as $C = \sqrt{\frac{\eta}{(BL \cdot \Delta w_{min})}}$.

In the same paper, they researched the requirements of a device to be able to give acceptable training results with this algorithm. It is important to mention that they found that a Δw_{min} smaller than 0.01 and an up/down device asymmetry of smaller than 1.05 are necessary. These restrictions have motivated the development of new training algorithms, as we describe one of them in the following.

Tiki - Taka

The Tiki - Taka algorithm tries to address the common and significant problem of asymmetrical device response. It was introduced by Tayfun and Haensch in [33]. There are two things that characterize this algorithm, one is a symmetry point shifting technique and the other is that the weight matrix is a linear combination of two matrices : $\mathbf{W} = \gamma\mathbf{A} + \mathbf{C}$. The Tiki - Taka algorithm is executed as follows:

1) Forward pass
2) Backward pass
3) Update **A**
4) After a selected number of previous cycles, Forward pass only using **A** and typically with a one-hot encoded vector as input.

5) Update **C**, using the error produced from the forward pass of step (4)

The outcome of this process involves transferring the accumulated gradients to the C matrix, increasing the likelihood of convergence in the correct direction due to the proper signs of the accumulated gradients. However, it is probable that the magnitudes of these gradients are underestimated, leading to slower conversion towards the optimal point.

Nonetheless, Gokmen and Haensch demonstrated in their research that this method yields favorable outcomes for asymmetric devices, thereby unlocking the potential of devices such as FTJs, which often exhibit asymmetry between positive and negative weight changes. In addition, new research shows better optimizations of this algorithm lead to reduced hardware and device requirements [34],[35].

2.3.4 Existing technology - Related Work

Multiple technologies have been explored for in-memory computing. Two main categories have been used, charge based, and resistance based. The first one uses transistors and capacitors to control the current intensity with widely used memory cells that are easily implementable on silicon. The second one uses materials that can change and maintain their electrical conductance depending on the electrical field applied so call memristors.

The charge-based memories have the advantage of reusing well known and robust architectures like SRAM (Static Random Access Memory), DRAM (Dynamic Random Access Memory) and flash. SRAM stores a bit value between 2 inverters connected back-to-back. It allows a fast writing and reading operations but requires to be connected to power to retain its value and takes relatively high area with its 6 transistors per cell. The DRAM is composed of a capacitor and a transistor. The bit value is stored in the capacitor and accessed by the transistor. The needed area is reduced compared to SRAM but the values have to be refreshed frequently due to leakage. These 2 technologies store 1 bit values and required to be powered to work. They can be used to perform binary operation and by extension addition and multiplication, the high degree of parallelism is an advantage for fast operations. The flash memory stores charges in the floating gate of a transistor. This allows to set multiple states in a single cell and retain it when the power is cut. This technology can be used for analog VMM operation. Mythic AI[1] have developed and commercialized a chip using this technology.

The resistive-based memory is field of research which undergoes active research. Many technologies using various materials are studied to propose a consistent way to store data in a compact way using a programmable resistive devices. These devices can be set to multiple states and retain the value over time. The resistive propriety is interesting for AIMC because it allows to perform fast and efficient MAC operation.

The PCM (Phase Change Memory) technology have been implemented for AIMC on chip by IBM and is used as default RPU cell in AIHWKIT. It exploits chalcogenide glass property to have 2 different states, amorphous and crystalline, that have significant different resistive values. The programming consists of heating the glass to change its crystalline form. Intermediary states can also be achieved but with the limitation of drifting over time. IBM presented in 2021 a 256×256 in-memory compute (IMC) HERMES [36] core designed and fabricated in 14nm CMOS with backend-integrated multi-level phase-change memory (PCM). The core can operate at frequency over 1 GHz which each RPU is composed of 8 transistors and 4 memristors where each half is respectively connected to positive and negative inputs to achieve signed weights.

[1] Mythic AI is a company that develop AIMC modules using flash memory Mythic AI website

Electrochemical Random Access Memory (ECRAM) is another promising technology for AIMC (analog in-memory computing). The resistance update in ECRAM is achieved by moving ions from an electrolyte to the channel located between two metal contacts. ECRAM operates as a tripole, meaning that the programming contact is separate from the read/write contacts. This technology has been demonstrated using various materials such as lithium ions, hydrogen ions and metal oxide. ECRAM is a non-volatile memory, exhibiting good symmetry and low power characteristics. Several companies and universities have developed and demonstrated the utilization of this technology [37] [38] [39].

Chapter 3

Methods

In this chapter, we will first present the simulator used for this work. The simulator serves as a platform to model the AIMC using memristor based crossbar arrays. Next, we will introduce the DNNs we selected. These DNNs are chosen for the variety of tasks they perform. Then, we will focus into the methodologies used for programming memristors. In this context, 'programming' refers to the process of assigning a specific resistance value to the memristors for the purpose of data encoding. Following the discussion on programming techniques, we will address the issue of non-idealities in memristor devices and present different mitigation techniques employed. Finally, we will outline the methodology used in this work to evaluate the performance of the FTJs.

By presenting the simulator, selected DNNs, programming techniques, mitigation of non-idealities , and methodology, this chapter aims to provide a comprehensive overview of the key aspects involved in the research conducted on the FTJ memristors.

3.1 The Simulation toolkit

Achieving in-memory computing for neural networks using RPU crossbar arrays presents an immense challenge. To navigate this complexity, IBM's Analog Hardware Acceleration Kit (AIHWKit) [40], an open-source toolkit, was employed. This toolkit can simulate analog crossbar arrays within the Pytorch framework, offering a highly practical and efficient solution. The core component of this system is the "analog tile", an abstraction capturing the computations performed on a crossbar array. These building blocks establish the integration of existing neural network modules with analog components, thereby helping in the creation of Analog Neural Networks (ANNs) using Pytorch.

While other software packages exist, such as NeuroSim [41] and RxNN [42], for simulating ML workloads on non-volatile memory elements, they lack the flexibility that comes with the integration into frameworks like Pytorch. The development of intricate neural network models becomes a significantly more streamlined process with these integrated frameworks.

Additionally, these alternate simulators often overlook certain critical facets, such as advanced algorithms and pulse update schemes, which are vital for training-enabled chip designs. IBM has also ingeniously devised a straightforward method for hardware-aware neural network training by simply excluding the non-idealities in backpropagation and weight updating processes. Overall this tool provides users a robust, efficient, and user-friendly simulator, providing a powerful tool for exploring the realms of in-memory computing and neural networks.

3.1.1 FTJ models and representation in the simulator

The FTJ device that was physically constructed at NanoLund exhibited significant access resistance, leading to a relatively large RC constant. This situation prevented the application of rapid pulses (<500 ns) to the device, and therefore the full potential of its transient response could not be thoroughly investigated. Furthermore, the high resistance of the device complicated the reading of the current at the output. However, these technical issues are anticipated to be addressed in future iterations, which would allow the FTJ model to be further refined.

Through a previously implemented compact model, the transient response of the FTJ devices was extrapolated to 1 ns pulse lengths and below, which allowed to define three distinct models of the FTJ device that were formulated within the simulator. The "Extreme" model is grounded purely in the measured device's characteristics at pulse lengths around 1 µs. The "Balanced" model corresponds to the extrapolated device behavior at ns pulse lengths, assuming the large access resistance issue was resolved. Finally, the "Constrained" model is based on the Balanced model but applies if device only operated in the middle range (constrained) of polarization values, where it exhibits a linear response to the pulses.

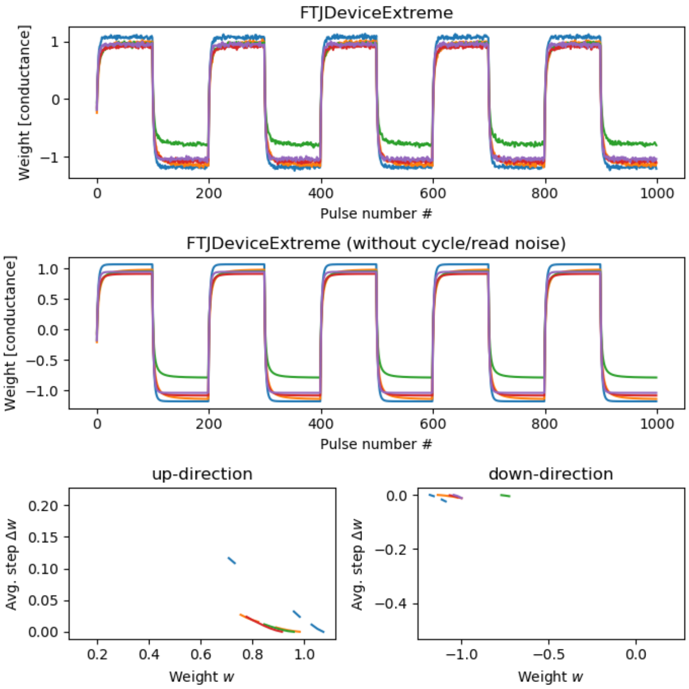


Figure 3.1: Response model of the "Extreme" FTJ device in the IBM AIHWKit simulator. Upper graph show cycle-to-cycle and device-to-device variations. Middle graph show device-to-device variations only. Last graph depict the average step based on weight value.

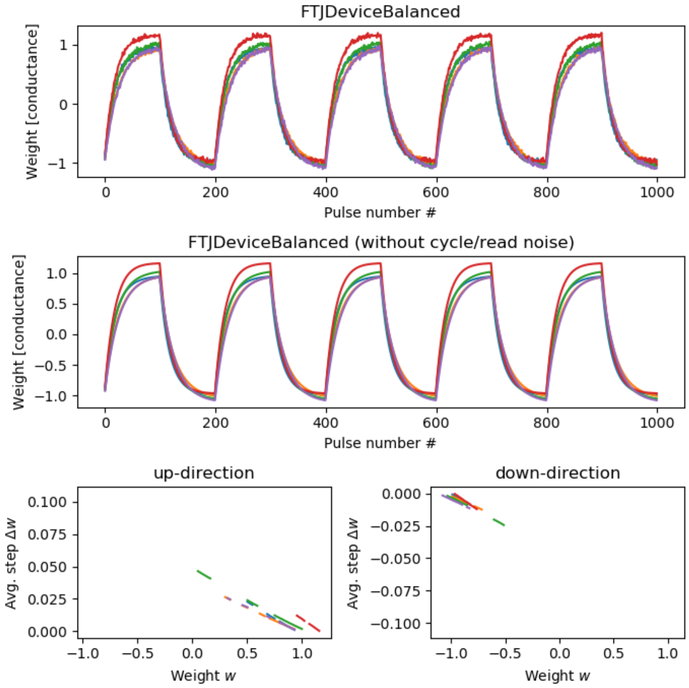


Figure 3.2: Response model of the "Balanced" FTJ device in the IBM AIHWKit simulator. Upper graph show cycle-to-cycle and device-to-device variations. Middle graph show device-to-device variations only. Last graph depict the average step based on weight value.

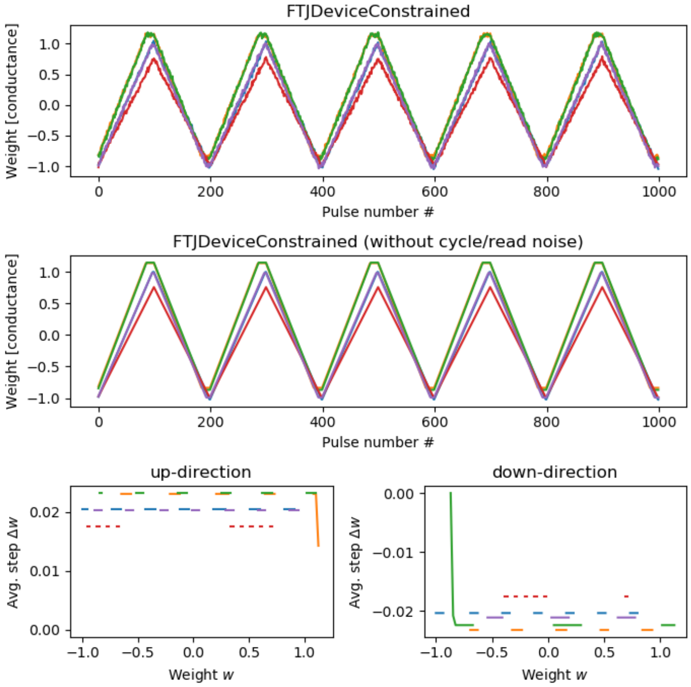


Figure 3.3: Response model of the "Constrained" FTJ device in the IBM AIHWKit simulator. Upper graph show cycle-to-cycle and device-to-device variations. Middle graph show device-to-device variations only. Last graph depict the average step based on weight value.

3.1.2 Tile model

The baseline hardware architecture for the tile which was simulated in the AIHWKit, has the following specifications (see Table 3.1). These parameters are selected according to a pessimistic approach to the available hardware. The ADC and DAC resolution is selected 8 bit in order to be comparable to the work of IBM [43]. The system noise referred to the output is of the range of one Least Significant Bit (LSB) of the ADC. As for the IR drop ratio, it is calculated by the ratio of the maximum memristor conductance to the conductance of the wire. The conductance of the wire is estimated around 0.35 Ohm for a typical technology (recommended by the simulator). Regarding the short-term weight noise, it was assumed that the effect of thermal and shot noise are the ones contributing to this. Their values are given by [44]:

$$I_t = \sqrt{\frac{k_B \cdot T \cdot BW}{R}} \tag{3.1}$$

where I_t is the thermal noise current, k_B ($1.38 \cdot 10^{-23}$ joules/K) is the Boltzman constant, T (K) is the temperature of the resistor, R the resistance and BW (Hz) is the bandwidth of operation. And,

$$I_{sh} = \sqrt{2 \cdot e \cdot I_{dc} \cdot BW} \tag{3.2}$$

where I_{sh} is the shot noise current, e ($1.6 \cdot 10^{-19}$ coulombs) is the electron charge and I_{dc} is the average dc current (A). Using the equations 3.1 and 3.2 for 10 MHz of inference operation, average resistance of 100 $M\Omega$, temperature 300 K and a measured averaged dc current of 1.78 nA and normalizing the result to the values between 0 to 1, the obtained noise is 3.83 %. For more optimistic results,the short-term weight noise was rounded to 3.5 %.

It is worth noting that the FTJ device exhibits multiple noise sources, which need further experimental modeling. However, the intensity of these noises depends on the device's size. Consequently, a new model should be created for varying technology nodes.

Table 3.1: Parameters of Baseline FTJ AIMC model

Parameter Description	Value	AIHWKIT configuration name
ADC precision	8 bit	forward.out_res
DAC precision	8 bit	forward.in_res
System noise referred to output	0.04	forward.out_noise
Output bound	10	forward.out_bound
Input bound	1	forward.in_bound
G-max	14nS	noise_model.g_max
Wire-conductance to gmax ratio	285714285.714	forward.ir_drop_g_ratio
Short-term weight noise	0.035	Additive Constant, forward.w_noise
IR-drop scale	1	forward.ir drop
FTJ drift	data calibrated	FTJLikeNoiseModel
Drift compensation	Global	GlobalDriftCompensation
Layer bias	digital	mapping.digital_bias
Digital output scale	collumn wise	mapping.out_scale_columnwise

3.2 Selected models

3.2.1 Simple Fully Connected Neural Network

A simple neural network consisting of 3 Layers and 235,146 trainable parameters was first used. The layers consist of a fully connected layer (784x256) followed by a Sigmoid activation feeding to another fully connected layer (256x128) with a Sigmoid activation again, and at last the output is processed by fully connected (256x10) and passed to the LogSoftmax for a probabilistic representation.

The digital FP32 model was trained using a learning rate of 0.1, a batch size of 64, and allowed to reach convergence within 25 epochs. For the analog model, the learning rate was halved to 0.05, due to the weight updates being influenced by the smallest step a device can take following an update pulse.

Classification of MNIST dataset

The Modified National Institute of Standards and Technology (MNIST) dataset [45] is a large database of handwritten digits that is commonly used for training various image processing systems. The dataset was created by Yann LeCun, Corinna Cortes, and Christopher Burges for evaluating machine learning models on the task of digit recognition.

Figure 3.4: Sample of MNIST dataset [45]

MNIST is often considered as the "Hello, World!" of machine learning, serving as a benchmark for classification algorithms. It consists of 70,000 images of handwritten digits from 0 to 9. These images are grayscale and have a size of 28x28 pixels. The dataset is divided into two subsets: a training set containing 60,000 examples, and a test set composed of 10,000 examples. Each image is labeled with the digit it represents. It is so simple that even small binary neural networks have been proven to classify properly this images. Currently the state of the art performance is 99.84 % accuracy.

3.2.2 ResNet-32

ResNet-32 is a model comprising 29 convolution layers and one fully connected layer, with 514026 trainable parameters, inspired by [46]. It is used for classification and in this case trained on the CIFAR-10 dataset. It is a model of intermediate size and it was used for evaluating both training and inference with RPU crossbar arrays. The convolution layers consist of kernels of size 3, padding 1 and stride 1. All of the layers are followed by batch normalization layers. The FC layer has a size of 64 by 10 since the dataset has 10 classes.

To train the FP32 version of the model, the hyperparameters were set as follows: learning rate 0.1, batch size 128, momentum 0.9, weight decay 0.0001 and trained for 200 epochs. Also a multi step function was used, that multiply the learning rate at 100 and 150 epoch by 0.1 of the original value. As for the dataset, augmentation techniques like random flip, crop and normalization of values with the mean and variance were used. The same was done for the AIMC counterpart but this time the learning rate was 0.01.

Classification of CIFAR-10 dataset

The Canadian Institute For Advanced Research (CIFAR-10) dataset is a widely used dataset for machine learning research. It was introduced by researchers at the CIFAR institute and has since become a standard dataset for testing machine learning algorithms, particularly in the field of image recognition.

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 different classes [47]. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.

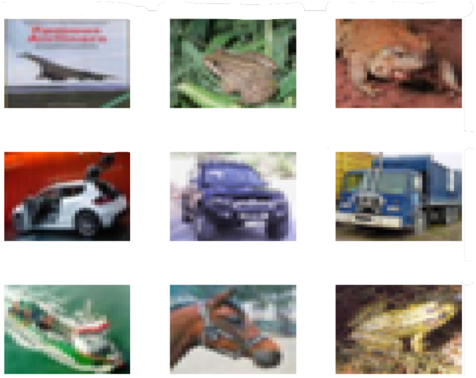


Figure 3.5: Sample of CIFAR-10 dataset

Similar to the MNIST dataset, CIFAR-10 is split into two subsets: a training set and a test set. The training set contains 50,000 images, while the test set contains 10,000 images.

The CIFAR-10 dataset serves as a good starting point for developing and practicing image recognition algorithms, especially CNNs. The dataset's small size makes it manageable, yet its complexity makes it challenging enough to draw reliable conclusions and insights.

3.2.3 YOLOv8

To demonstrate the capabilities of the FTJ in analog in-memory computing for image segmentation, we choose to use YOLOv8 fully convolutional neural network. This is a state-of-the-art model released on January 10th, 2023. It can be used for segmentation, detection, classification and pose detection tasks.

YOLO stand for You Look Only Once, the model performs detection on the whole input image in only one forward pass which make it fast compared to the previous model types used for the same purpose. The model is anchor free, which means it doesn't rely on predefined boxes to detect an object. To achieve this, the input image is divided into a grid of cells, where each cell is responsible for detecting the objects located in it. Each cell predicts objectness scores, class probabilities, and geometrical offsets to estimate the bounding box of the object. The geometrical offsets are relative to the cell center allowing to localize objects without relying on predefined anchors or reference points. To remove the overlapping detection of the same object, a non-maximum suppression algorithm is applied at the end.

YOLOv8 is develop by Ultralytics [1] which proposes the model in 5 different sizes for each task. For our work, we will focus on the segmentation and detection models with the size 'n', for nano, which is the smallest option (3.2 million parameters compared to 'x', for extra large, the biggest one with 68.8 million parameters for the detection model). The nano model has a lower accuracy but segmentation still gives acceptable results as shown if Figure 3.6. This option has been selected for its limited size which allows an analog hardware-aware retraining but still gives good visual results. The input images are RGB 640x640 pixels. The retraining in analog domain is a bottleneck in this work because it is resource hungry and requires multiple hours per epoch to train with the AIHWKit API and the complete COCO dataset.

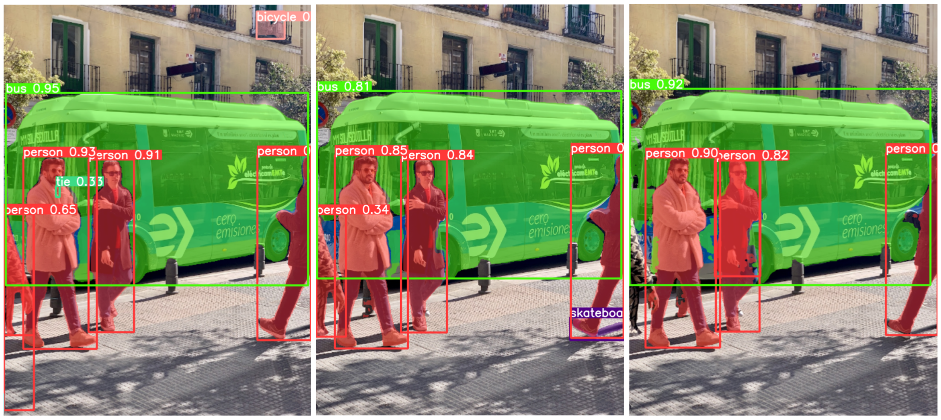


Figure 3.6: Different precision models for YOLOv8. From left to right: x, n, n converted to analog.

The main difference between the detection and segmentation models is that the segmentation one uses a deconvolution module to upsample the feature maps to match the size of the input image. Then the upsampled output goes into a Softmax layer to generate the probability distribution over the semantic labels of the input image. The model is pre-trained on the COCO 2017 dataset. COCO [2] stands for Common Objects in COntext, It's a widely used dataset in computer vision research and specifically in the fields of object detection, segmentation and captioning. It was created by Microsoft and provides a comprehensive collection of images with complex scenes and diverse object categories. The version we are using, provided by Ultralytics, have the following split: training, validation and test sets contain respectively 118287, 5000, 20288 images and 80 classes. The dataset includes detailed annotations such as bounding boxes, segmentation masks, and captions, enabling us to tackle various computer vision tasks. COCO's interest lies in its ability to address real-world challenges by capturing objects in their contextual scenes and offering a rich and diverse range of visual data. A quality dataset like COCO is essential to expect proper segmentation results.

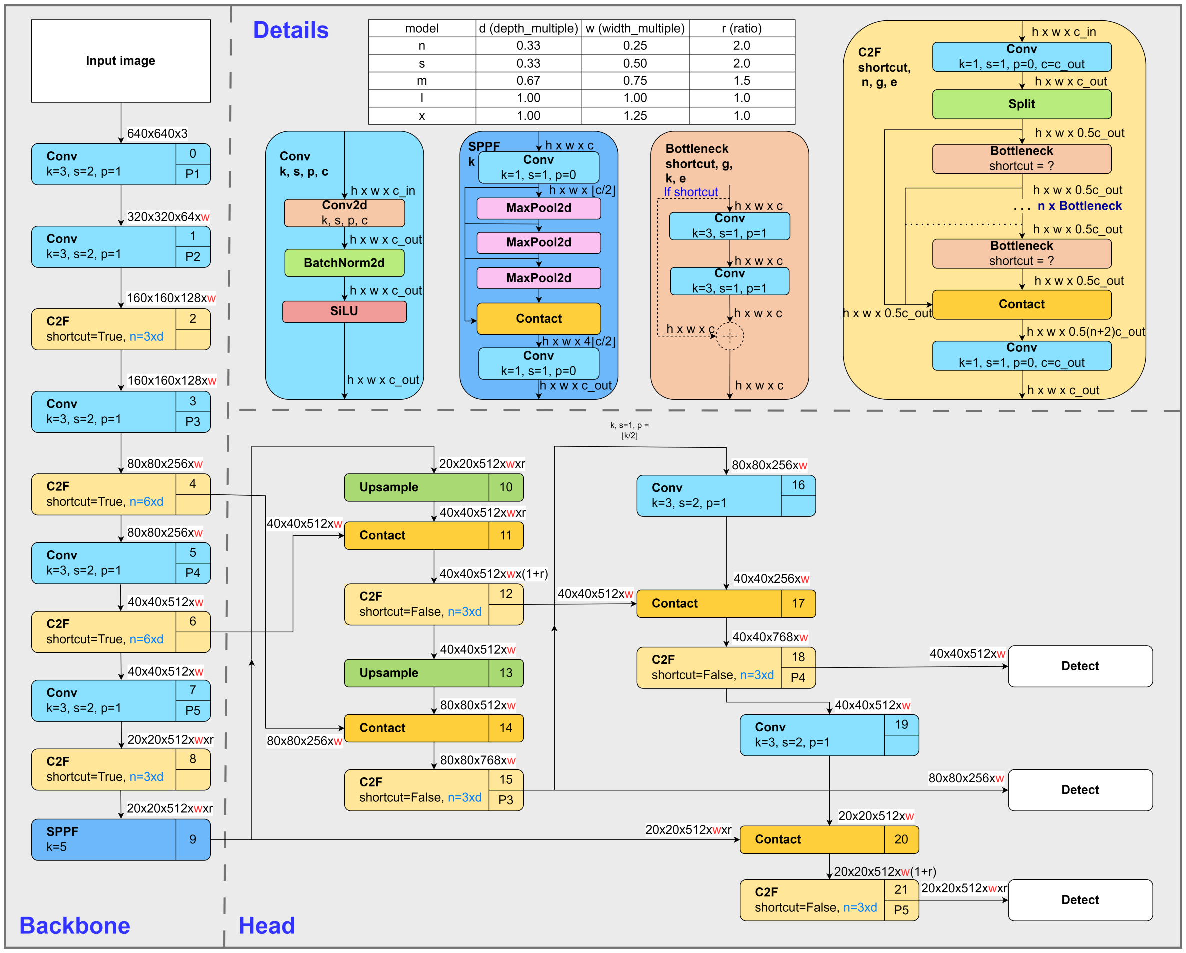
One of the key points for YOLOv8 efficiency is the usage of data augmentation and notably the mosaic. It expands the effective size of the training dataset by combining multiple images into a single mosaic. It enables the model to learn from a diverse range of object configurations

[1] Ultralytics is the official provider of YOLOv8 Ultralytics website

[2] Google COCO 2017 dataset COCO website

and backgrounds, thereby increasing training data efficiency, improve the generalization and reduce the overfitting risk.

To compare our results, we are using the metric mAP50 standing for mean Average Precision at 50% intersection over union (IoU). It's widely used as evaluation metrics for detection and segmentation tasks. It measures the accuracy of object detection and segmentation algorithms by evaluating how well they can localize and classify objects in an image. Average precision (AP) was introduced as precision metric in the paper "The PASCAL Visual Object Classes (VOC) Challenge" [48]. The mAP is the average of the AP computed for each class of the model.



model	d (depth_multiple)	w (width_multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Figure 3.7: YOLOv8 architecture for detection model.

3.2.4 BERT

Bidirectional Encoder Representation from Transformers also known as BERT is a type of Transformer architecture that only takes advantage of the encoder part. In other words, it is a stack of the encoder architecture which is capable of understanding human language and context. Bert is usually deployed for Natural Language Processing (NLP) tasks like translation, question answering, sentiment analysis and text summarising. There are variations of the BERT architecture like small or big. In this work, the BERT-base version was used. It has 12 encoder stacks and an FC output layer 3.8, pretrained by HuggingFace on Squad v1.1

dataset. The number of FC layers in the encoder stack is 72 plus one of the output, a total of 73 FC layers with 109,096,136 trainable parameters.

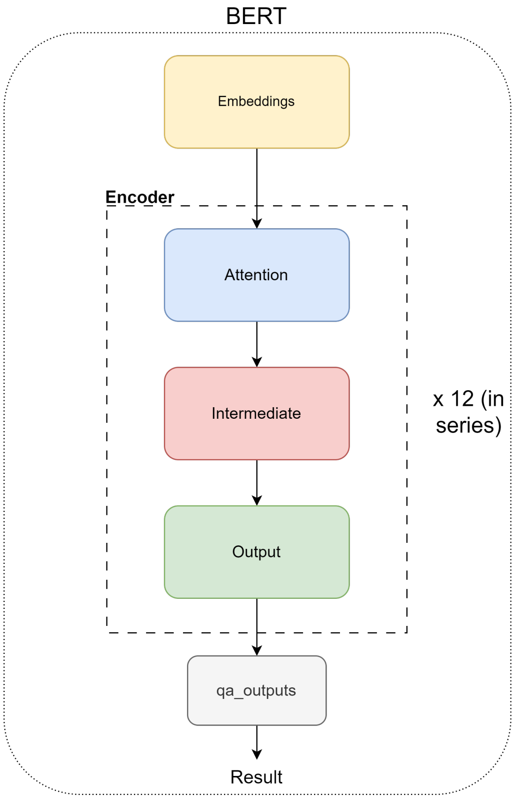
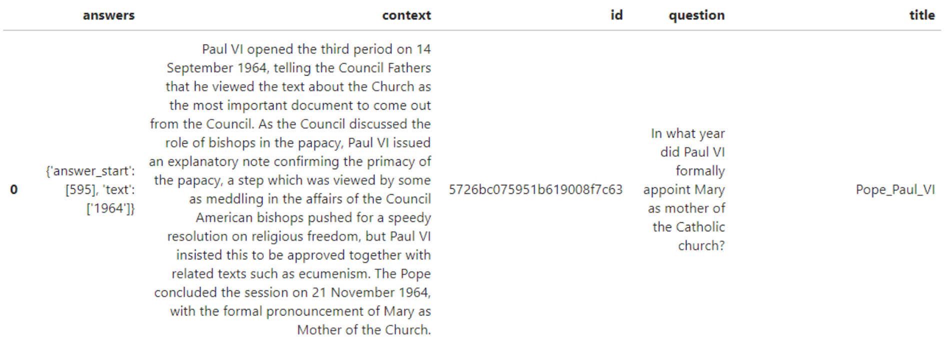


Figure 3.8: Higher level representation of the BERT architecture. The main blocks are shown. Inside the blocks are FC layers and word embedding algorithms.

This model was used only for evaluation of the inference capabilities of the AIMC, due to the extensive data power and time which would be required to train it.

Q&A on SQuAD v1.1 dataset

The Stanford Question Answering Dataset (SQuAD) v1.1 is a widely used dataset in the field of Natural Language Processing (NLP). It was developed by researchers at Stanford University to facilitate research in the development of models capable of understanding and answering questions based on a given context [49]. The dataset consists of 107,785 questions asked by crowdworkers on a set of Wikipedia articles, where the answers are segments of text from the corresponding reading passage, e.g. 3.9.



	answers	context	id	question	title
0	{'answer_start': [595], 'text': ['1964']}	Paul VI opened the third period on 14 September 1964, telling the Council Fathers that he viewed the text about the Church as the most important document to come out from the Council. As the Council discussed the role of bishops in the papacy, Paul VI issued an explanatory note confirming the primacy of the papacy, a step which was viewed by some as meddling in the affairs of the Council American bishops pushed for a speedy resolution on religious freedom, but Paul VI insisted this to be approved together with related texts such as ecumenism. The Pope concluded the session on 21 November 1964, with the formal pronouncement of Mary as Mother of the Church.	5726bc075951b619008f7c63	In what year did Paul VI formally appoint Mary as mother of the Catholic church?	Pope_Paul_VI

Figure 3.9: Sample of SQuAD dataset

The metrics used to indicate how successful a network is at answering these questions are the F1 [50] and Exact Match (EM) ones. The concept of Exact Match (EM) scoring is quite straightforward. In the context of each question and answer pair, the EM score is determined by the character-for-character accuracy of the model's prediction in comparison to the actual answer. If the model's predicted answer is perfect match with one of the correct answers, the EM score is assigned as 1. However, if there is even a single character discrepancy, the EM score drops to 0. This metric is quite strict. Furthermore, in the case of a negative example, if the model predicts any text, it is automatically assigned an EM score of 0 for that particular instance. Meanwhile, the F1 score is calculated based on the individual words in the predicted answer compared to the actual answer. The calculation of the F1 score is grounded in the number of words that the prediction and the actual answer have in common.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

where precision is the ratio of the number of shared words to the total number of words in the prediction, and recall is the ratio of the number of shared words to the total number of words in the ground truth. This metric holds greater significance because an answer can be provided using different words.

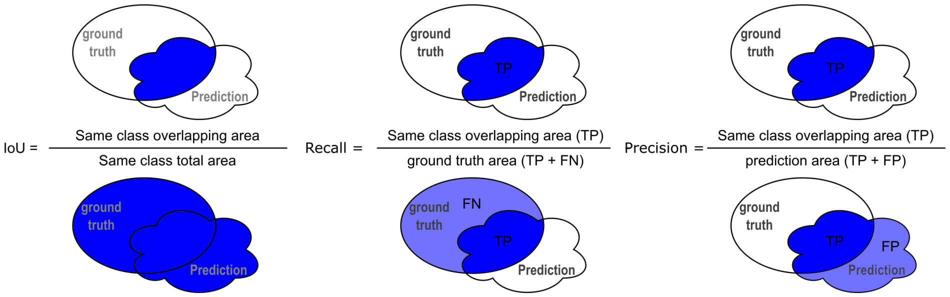

ground truth
Prediction
IoU =
Same class overlapping area
Same class total area
ground truth
TP
Prediction
Recall =
Same class overlapping area (TP)
ground truth area (TP + FN)
FN
ground truth
TP
Prediction
Precision =
Same class overlapping area (TP)
prediction area (TP + FP)
FP

Figure 3.10: Metrics visualisation for intersection over union (IoU), recall and precision.

3.3 Weight programming

The process of programming a memristor with precision to achieve a desired conductance value represents a considerable challenge. Various programming schemes are available, each possessing specific advantages and disadvantages. However, the primary consideration in scheme selection is the level of accuracy that can be attained. This section elaborates on the programming schemes experimented with during this study and proposes areas for potential further exploration.

It's important to clarify that the IBM simulator doesn't support variable amplitude pulsing schemes; it's designed to handle device responses based on pulses of a uniform amplitude. This restriction means that the models developed within the simulator don't fully represent the capabilities of the FTJ device. To simulate responses to variable amplitude, custom functions were incorporated into the simulator.

3.3.1 Tile Gradient Descent Programming (tGDP)

The tGDP scheme, short for "Tile Gradient Descent Programming," utilizes the methodology proposed in [23]. Instead of adopting a training-based approach, this scheme employs Gradient Descent Programming (GDP) on a per-tile basis. Each tile performs a forward pass, utilizing an Identity matrix (**I**) of corresponding dimensions as its input.This forward pass yields a result, **W'**, which comprises the weight matrix programmed on the crossbar array along with all its non-idealities. The error is then computed as $\mathbf{E} = \mathbf{W} - \mathbf{W'}$, where **W** is the ideal weight matrix. As elaborated in Section 2.3.3, both the input and error are converted into stochastic bit streams, thereby triggering the weight update cycle as illustrated in 3.11. Whenever pulses coincide on a memristor, the weight is modified by its smallest quantity, Δw.

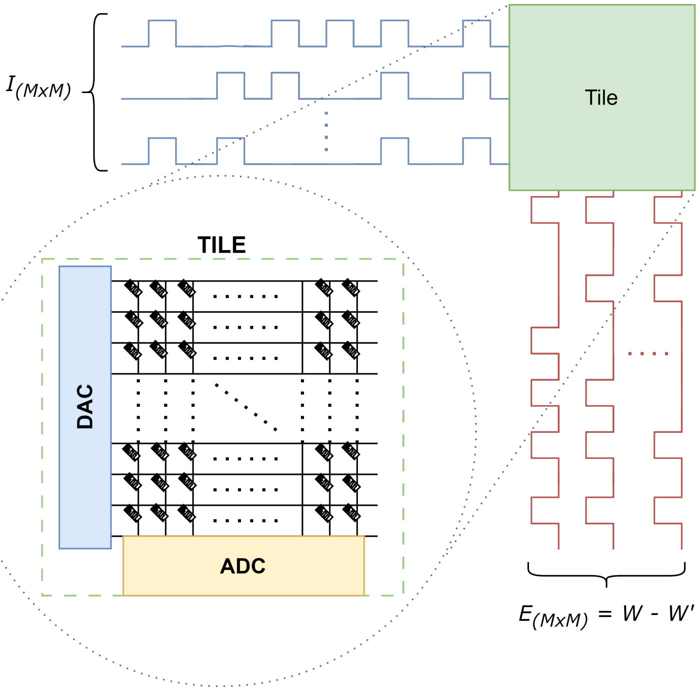


Figure 3.11: tGDP applied per Tile.

The outcome of this method is influenced by several hyperparameters. For instance, the update algorithm may be either deterministic or stochastic. Regardless of the chosen approach, the update pulses rely on several factors including the learning rate established for the tile, the number of memristors per RPU, the RPU's update policy, the designated maximum length of pulses, and the update management specifying the dynamic or static length of pulse trains. Moreover, the quantity of forward passes and a dynamic halt of programming iterations can be determined.

Within the context of this thesis, the hyperparameters were configured as follows:

Table 3.2: Parameters of Tile's Weight Updates

Parameter Description	AIHWKIT configuration name	value
Desired pulse train length	update.desired_bl	31
Pulse generator	update.pulse_type	STOCHASTIC_COMPRESSED
Fix pulse trains length	update.fixed_bl	True
Dynamical adjustment of A,B,and BL	update.update_bl_management	True
Apply additional scaling	update.update_management	True

3.3.2 Varying Pulse Approach

While the simulator does not fully support this approach, a Python function was developed, utilizing data derived from the work in [14]. This function effectively maps the weights to the normalized values attained via the Varying Amplitude pulsing method, as illustrated in Figure 3.12. Additionally, cycle-to-cycle and device-to-device variations are incorporated in this mapping process.

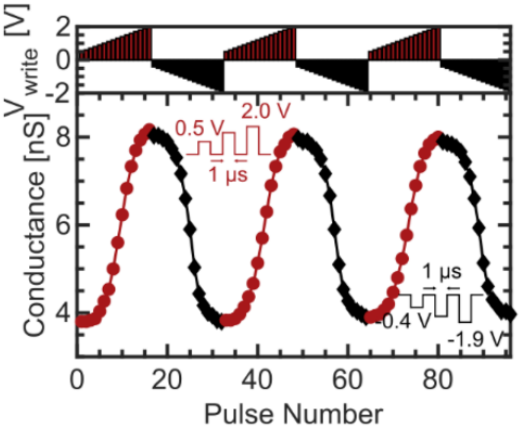


Figure 3.12: Varying pulse amplitude response. Source[14]

Figure 3.12 illustrates the case that the pulse duration is set to 1 microsecond for the FTJs fabricated at LTH. The voltage values range between 0.5 and 2 for potentiation, while for depression, they span from -0.4 to -1.9. This configuration generates 16 distinct conductance states per memristor. However, to accommodate negative values at each crosspoint, an additional memristor is required. This adjustment results in a total of 31 conductance states per RPU.

This method was also evaluated for multiple devices per crosspoint. However, the additional conductance states that were made available were significantly overshadowed by the effect of device-to-device and cycle-to-cycle variations. Consequently, a different approach was pursued.

3.3.3 X Binary - Single Analog Approach

The idea here is to mitigate the effect of the variations by reducing the importance of the multi-level memristor. Employing a certain number (X) of memristors in binary mode facilitates the assignment of the Most Significant Bits (MSB) to reliable states, thereby enhancing the Signal to Noise ratio (SNR). Meanwhile, the small residual part which is calculated by the multilevel operated memristor can cover the least significant values. In the case of representing a weight, such as 0.80, and assuming there are 2 memristors in binary mode along with 1 in analog mode, the process would proceed as follows: The 0.80 is converted into an 8-bit binary, resulting in 0.11001100. Subsequently, the two binary mode memristors are set to high conductance, while the remaining value is programmed onto the analog memristor.

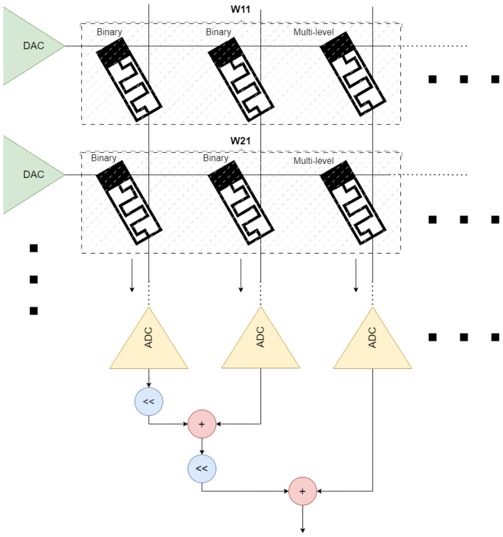


Figure 3.13: Two binary One Multi-level

Combining this method with the weight scaling technique which will be introduced later, the overall noise is scaled further down. However, the implementation of this demands extra area, power and peripheral circuitry (especially ADC). The actual hardware implementation of this needs further exploration, since the scope of this thesis is focused on functional accuracy. The Figure 3.13, showcases the RPU crossbar structure in case of two binary and one multi-level memristors.

3.4 Mitigation of non-idealities

To mitigate the various non-idealities, a variety of approaches have been introduced, each one contributing to the outcome of the evaluation.

3.4.1 Weight Scaling and Clipping

In this case the weight matrix that is programmed on the crossbar array can be in some cases clipped at a chosen deviation of the distribution of the matrix and then scaled by a factor $\gamma = 1/w_{max}$, in other words normalized. In this way the distribution of weights is extended and the whole range of possible resistive states is taken advantage of. Moreover, this adjustment leads to an increase in current because of the elevated conductance values. This increase effectively the signal and moves it above the noise floor leading to an increased SNR. This scaling can also be done per column, which is even more effective. However, the process needs extra peripheral components that re-scale the outcome and also calculate the maximum of each column. Nevertheless, this technique has been proven of utmost importance in achieving iso-accuracy [43].

3.4.2 Noise and Bound management

In this work, the bound and noise management mitigation techniques implemented in the AIHWkit simulator have been used. These 2 techniques use an α vector to modify the input range in order to use the maximum range of the output and to move away from the noise floor. The noise management determines a factor α to maximize the input range by normalizing it to the maximum input value for each inference. In our case, the normalization uses the absolute maximum value of the inputs. The following equations are used:

$$F(y) = \alpha \, F_{\text{analog-mac}}\left(\mathbf{x}/\alpha\right) \text{ for } \alpha = \max|\mathbf{x}| \tag{3.3}$$

This techniques allow to use the maximum range of the input and increase the SNR generated for small inputs. The main drawback of this technique is the need of hardware to determine the maximum value of the inputs, divide each input and then multiply the output.

The bound management iteratively divide by 2 the previously computed α if the output ADC is saturated. This technique allows to use the full dynamic range of the ADC but requires some extra hardware to detect the overflow and update the α vector. It also impacts the throughput by iteratively recomputing until no more saturation is detected, that removes the deterministic computing time.

These techniques optimize the usage of the crossbar array by using the full range of input and output but require extra hardware and extra time for computation. To remove the need of them, the α vector can be trained as it was previously proposed [43]. In this study, it was not trained due to the hardware limitation and time needed to do it.

3.4.3 Drift Compensation

Conductance drift in the memristor device hardware can be very detrimental on the inference performance of a model mapped to hardware. The weights lose their initial values and as a result the activation of neurons is not proper any more. In the case of the FTJ device, experimental data were used to model the drift effect, as shown in 3.14. The conductance evolution is empirically modeled as:

$$g_{drift}(t) = g_{prog} \frac{t}{t_c}^{-v} \tag{3.4}$$

where v is the so-called drift exponent. The drift exponent exhibits a dependence on the target conductance given. Due to the lack of multiple measurements, the equation that gives the drift exponent was made deterministic and is given as:

$$v = 0.0122|w|^3 - 0.0241w^2 + 0.0112|w| + 0.001 \tag{3.5}$$

where w is the weight.

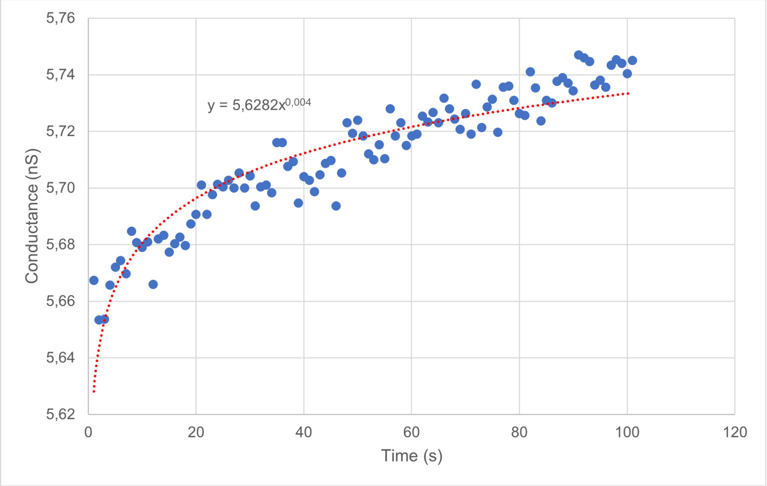


Figure 3.14: a subset of the experimental data that illustrates the change in one resistive state of the FTJ over a span of 100 seconds. The red line is the fitted curve based on the Equation 3.4 which was used to model the drift in the simulator.

Despite the good state retention of the FTJ, we suggest a simple mitigation method for further improvement. This is because, in an array setup, each device's drift accumulates and contributes to the overall error. A technique that counters the effect of drift to a certain degree was proposed in [51]. In this case a global calibration process is used. Periodically, a subset (L) of the columns in the array is read using a constant voltage V_{cal}. Then a scalar α is calculated as:

$$\alpha = \frac{I_{cal}}{V_{cal} \sum_{n=1}^{N} \sum_{m=1}^{L} G_{m,n}(t_0)} \tag{3.6}$$

where I_{cal} is the current produced by the prementioned read and $G_{m,n}(t_0)$ is the inital conductance values. This means can be expressed as well as:

$$\alpha = \frac{I_{cal}(t)}{I_{cal}(t_0)} \tag{3.7}$$

With this α scalar it is possible to rescale the matrix vector multiplication result by $1/\alpha$, recalibrating the outcome to be closer to the appropriate one.

3.4.4 Hardware Aware Training

The transition from digitally computed models to analog is not as trivial as simply mapping the parameters to conductances. Analog devices introduce various non-idealities associated with the hardware, which, if not carefully addressed, can result in significant accuracy degradation. However, it has been demonstrated that reduced-precision digital accelerators, such as the Google TPU [52], have a notable capacity to cope with substantial precision reduction in parameters resulting from quantization. It has been shown that retraining DNNs in a quantization-aware manner yields acceptable accuracy even when using a weight representation of only 4 bits. A similar approach can be employed to enhance the accuracy of analog models.

Analog computation is inherently limited by the hardware used. The DAC precision limits the input range, memristor technology limits the number of states that can be used, and the ADC limits the precision of the result. Additionally, electronic and thermal noises, along with variations between devices, further contribute to non-deterministic behavior, leading to significant mismatches during the reading and writing of conductances. Mitigation techniques can be applied to limit these effects, such as input or synaptic bit slicing [22], which respectively remove the DAC and increase the achievable states by using multiple memristors. However, these stochastic non-idealities affect the computation of DNNs differently than the deterministic weight quantization used in digital models. Therefore, new retraining techniques must be applied to achieve proper accuracy.

To retrain multiple models, we chose to follow the procedure proposed in [43]. The retraining involves applying all the expected non-idealities and noise levels during the forward path. The backward and update parts are performed in FP32 precision to apply only the non-idealities seen during inference. All on-chip and chip-in-the-loop retraining have been carefully avoided, even though they can achieve more accurate results, we wanted to ensure that the models are generalizable and perform well on different chips. The hardware-aware retraining starts from a digital model that we trained or by using the best available weights. Then, the hyper-parameters are adjusted to optimize the retraining. The adjustability of the hyperparameters is limited by the resources used in terms of memory and time.

The base RPU used for this work is similar to the one proposed by Rasch, M.J., Mackin et al.[43] (Table 3.1), where the PCM specific parameters have been replaced with the 'Balanced' FTJ parameters, and the input range is not trained. Instead, it is managed by noise and bound management techniques 3.4.2. This choice was made to speed up and facilitate the process. The 'Balanced' preset and the FTJ noise model have been used to tune the retraining.

During this work, attention have been made to follow as much as possible the proposed method [43] to allow comparison.

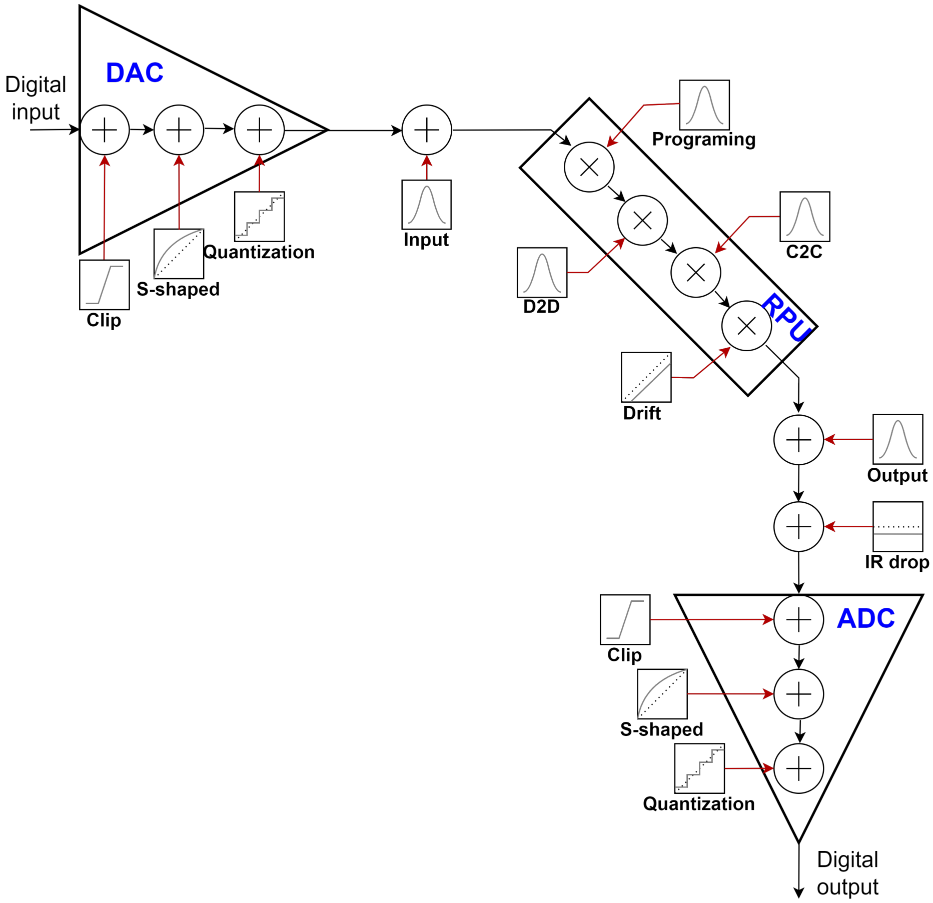


Figure 3.15: Inference noises used during hardware-aware training

HWA ResNet32

Following the previously discussed plan, the ResNet model was the first to undergo testing with this approach. The learning rate was set at 0.015, with momentum at 0.9 and weight decay at 0.0001 for the Analog SGD optimizer. The batch size was maintained at 128, and the model was trained for 200 epochs. During this training period, noise similar to that found in the Balanced device response was introduced, but at three times the original strength.

HWA BERT-base

For the BERT model, an Analog SGD optimizer with a learning rate of 0.01 and a weight decay of 0.001 was used. The batch size was set at 5, and the model was trained for 15 epochs. During the forward pass, noise was introduced that was six times stronger than the noise used with the ResNet model. Additionally, a very small amount of input noise was added at each analog layer.

HWA Yolov8-nano

For this work, YOLOv8 for segmentation has been converted to analog and retrained in an hardware-aware manner using the baseline RPU (see Table 3.1). As mentioned in the [43], retraining CNNs is quite challenging and have a tendency to exhibit more pronounced accuracy drops.

The 'nano' version of YOLOv8, which has 3.2M parameters, has been retrained with a smaller subset of the COCO dataset to reduce the epoch time. These subsets have been generated by randomly selecting 10240 unique images of the COCO training set for training, and 1024 unique images of the validation set for validation. Each image is RGB and has a size of 640x640 pixels. This relatively high-resolution increases the memory and the processing time needed for the computation.

The model have been retrained using the 'best weights' provided by Ultralytics and the model 'yolov8n-seg.pt' where each Conv2d have been converted to AnalogConv2d except the DFL layer. The following hyperparameters have been used: epoch: 60, patience: 50, optimizer: AdamW, lr0: 0.0004375, lrf: 0.05, momentum: 0.937, weight_decay: 0.0005, warmup_epochs: 0.0 and a batch size of 8. The best epoch was the 47th.

The rather small batch size was forced by hardware limitation. Even with the usage of two Nvidia RTX 3090, each equipped with 24GB of memory, and a code optimization, the GPU memory was the bottleneck. Using the AIHWkit API increase dramatically the need of GPU memory to compute all the non-idealities.

The Distribution Focal Loss (DFL) convolution layer was kept in digital and its weights were frozen, i.e. exempted from retraining. It's a variation of the Focal Loss function that is used in YOLOv8 for the localization loss. The DFL is used to address the problem of class imbalance in the training data. To do so, the predicted geometrical features (x, y, w, h) are discretized into bins, using a predefined number of steps, and used to create a probability distribution across the range of possible values for each feature. The DFL aims at giving more weight to hard-to-classify examples.

The retraining is using a small subset of COCO, approx. 10%, doesn't show overfitting signs and seems to have some margin for improvement with more epochs. The mAP50 obtained with the retrained model after a 1 hour drift is 0.411 compare to the 0.495 of the digital model on the 1K validation dataset. The retrained model has a 83.1% accuracy compared to the digital. This might be increased by retraining for more epochs and keeping the first and last layers of the model in the digital domain.

The Figure 3.16 shows the hardware-aware re-training data of the yolov8 model for segmentation we used for the study. The left half part of the graph shows the result of the losses functions for each epoch. The top part is the training data, and the bottom part is the validation data. The right half shows the metrics computed for each epoch. They are separated in 2 groups, the 'B', for box, are the results obtained for the bounding boxes, it's equivalent to the detection task. And the 'M', for mask, are the results for the segmentation masks. The metrics used in this study is the 'metric/mAP50(M)'.

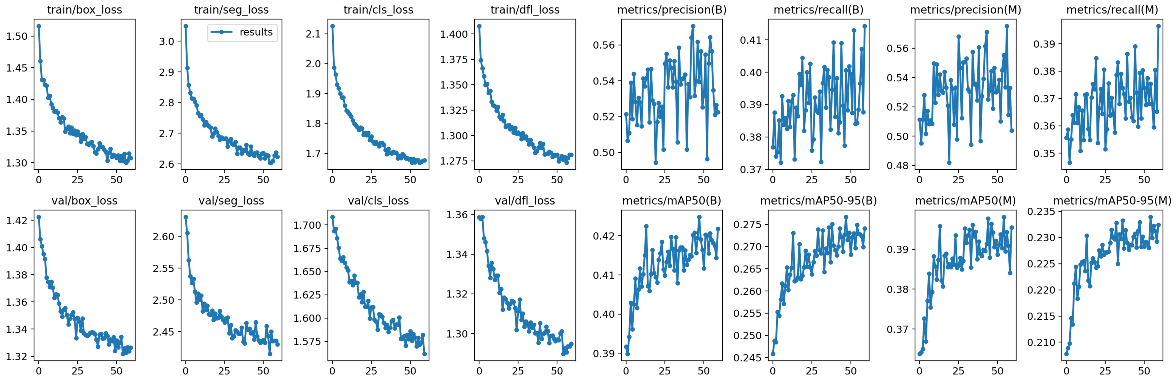


Figure 3.16: YOLOv8-seg hardware-aware retraining data for 60 epochs

3.5 Methodology

After selecting relevant models for our study, the following methodology has been applied to get a normalized way to compare performances along the analog converted models. The first step consists of performing the inference of the trained model in digital domain. The result gives us a reference to compare the performance of the analog in-memory model. A second inference is done in digital domain with weights randomly set. This allows to consider the default model performance for a fairer comparison. To do so, we are using the method proposed by [43].

$$A_*^{1h} = 1 - \frac{\epsilon_{test}^{1h} - \epsilon_{test}^{FP}}{\epsilon_{chance} - \epsilon_{test}^{FP}} \tag{3.8}$$

This method gives a normalized metric to compare analog to digital models for various neural network topologies. The A_*^{1h} represents the accuracy percentage of the analog model result compared to the digital one after 1 hour. ϵ_{test}^{FP} represents the floating-point result of the current experiment, ϵ_{chance} represents the floating-point result when the weights are randomly sets. ϵ_{test}^{1h} is the result of the current analog result after 1h drift. If the $A_*^{1h} = 1$, it means the analog model performs as good as the digital one. If it equals to 0, the analog model is performing as good as the randomly set weights.

The digital model is then converted into analog using the baseline RPU configuration (Table 3.1). To assess the model's behavior under realistic conditions, it is programmed onto AIMC, and one inference is executed to evaluate both 1-hour and 1-day drift scenarios. The experiment is repeated 10 times per drift scenario to obtain statistical results. These results are then used as a reference for the retraining.

The digital model is retrained in a hardware-aware manner to mitigate the effects of non-idealities and noise introduced by the analog conversion. To ensure proper accuracy across diverse hardware, the retraining procedure deliberately excludes device or chip-specific characteristics, aiming for a more generic model. The objective of this retraining is to achieve the digital ISO accuracy. The model undergoes the same programming/inference cycles for 1-hour and 1-day drifts to quantify the gain.

Once the model is retrained, a layer-to-layer analysis is performed. This provides insights into which layers are the most sensitive and require more attention. These layers can be retrained more carefully, the weights can be programmed more precisely, or they can eventually be kept in the digital domain.

In the last phase, the model's robustness is evaluated through a sensitivity analysis. This analysis involves varying the intensity of individual noise sources and non-idealities while keeping the rest of the RPU baseline unchanged. By conducting this analysis, valuable insights are obtained regarding the design and acceptable margins. The ultimate objective is to identify which DNNs are more suitable for AIMC and determine the sensitive parameters. For this study, the following parameters are tested:

Table 3.3: Parameters individually modified for sensitivity analysis

Parameter Description	AIHWKIT configuration name	Parameter variation
DAC precision	forward.in_res	4 to 8 bits
ADC precision	forward.out_res	4 to 8 bits
System noise referred to output	forward.out_noise	0% to 40%
Weight programming noise	forward.w_noise	0% to 40%

Chapter 4

Results

In this chapter, we will present the results obtained from our research. We will begin by presenting the usage of FTJs for training neural networks.After that, we will focus on the results obtained from the inference phase, which was the main part of our work.

By presenting the results obtained from training and inferencing neural networks using FTJs, this chapter aims to provide a thorough assessment of the capabilities and constraints of using FTJs in AIMC.

4.1 Neural Network Training Using FTJ RPU Crossbar Arrays

The efficacy of utilizing FTJ RPU crossbar arrays for training neural network algorithms was evaluated through experiments with two distinct neural network models, varying in complexity. These models were subjected to numerous RPU configurations during training.

The first neural network under consideration is a straightforward model comprising three fully connected layers and was trained using the MNIST dataset. The chosen hyperparameters for this model are the same as mentioned in the models section, apart from the learning rate that was half of the FP32 model. Training was conducted utilizing two algorithms: Analog SGD and Tiki-Taka (v2). Figure 4.1 shows the progress of the test error after each epoch for the various RPU configurations tested. The FP32 accuracy was 96% and only the RPU with the "Constrained" FTJ model managed to reach it.

The next training test was on the aforementioned ResNet-32 on CIFAR-10 dataset. Given the demonstrated shortcomings of the Extreme FTJ model, the training on this more intricate network was conducted using more advanced and effective RPU configurations. The hyperparameters for this are the same as in models section but the learning rate was divided by 10 compared to the FP32. The relative performance of each configuration, benchmarked against an ideal model, is presented in Figure 4.2. Some of the results were stopped earlier since there was no progress in convergence. The FP32 accuracy was 91.75%, a score that none of the AIMC models managed.

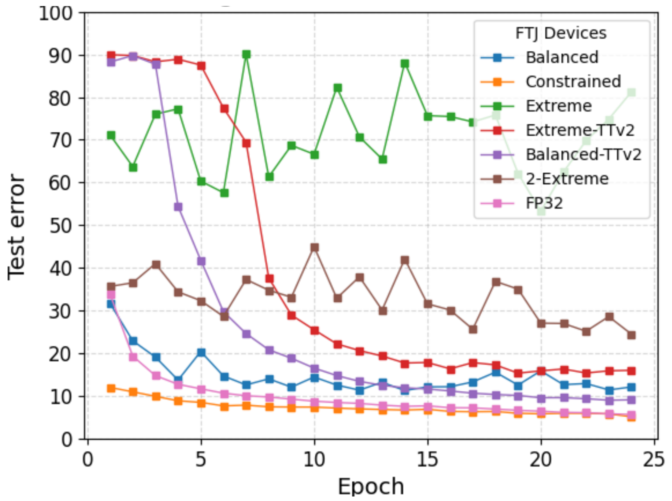


Figure 4.1: Training of a neural network consisting of three fully connected layers on MNIST dataset, using RPU crossbar arrays via either the Analog SGD or Tiki-Taka (v2) algorithm. Various memristor devices are employed as RPUs in different scenarios, as indicated in the legend. Each case is compared against a model trained on a 32-bit floating point device (FP32). The numerical prefix associated with each device represents the quantity of such devices per weight. A suffix of 'TTv2' denotes a device specifically configured for the Tiki-Taka algorithm.

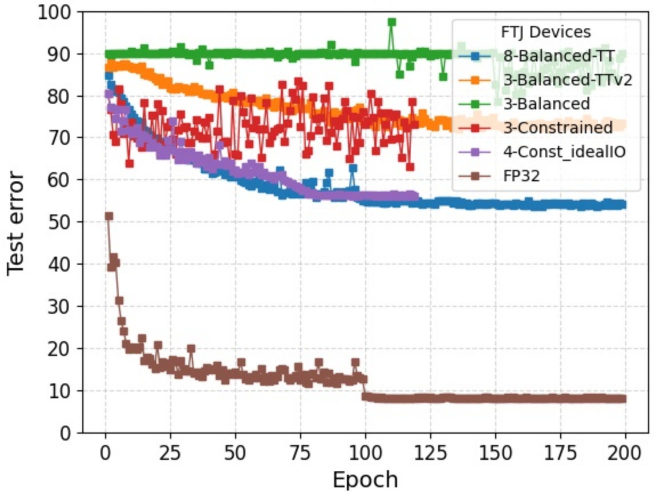


Figure 4.2: Training of ResNet-32 on CIFAR-10 dataset, using RPU crossbar arrays via either the Analog SGD or Tiki-Taka (v2) Algorithm. Various memristor devices are employed as RPUs in different scenarios, as indicated in the legend. Each case is compared against a model trained on a 32-bit floating point device (FP32). The numerical prefix associated with each device represents the quantity of such devices per weight. The suffixes 'TT' and 'TTv2' denote a device specifically configured for the Tiki-Taka algorithm version 1 and 2 accordingly.

4.2 Neural Network Inference Using FTJ RPU Crossbar Arrays

4.2.1 VMM

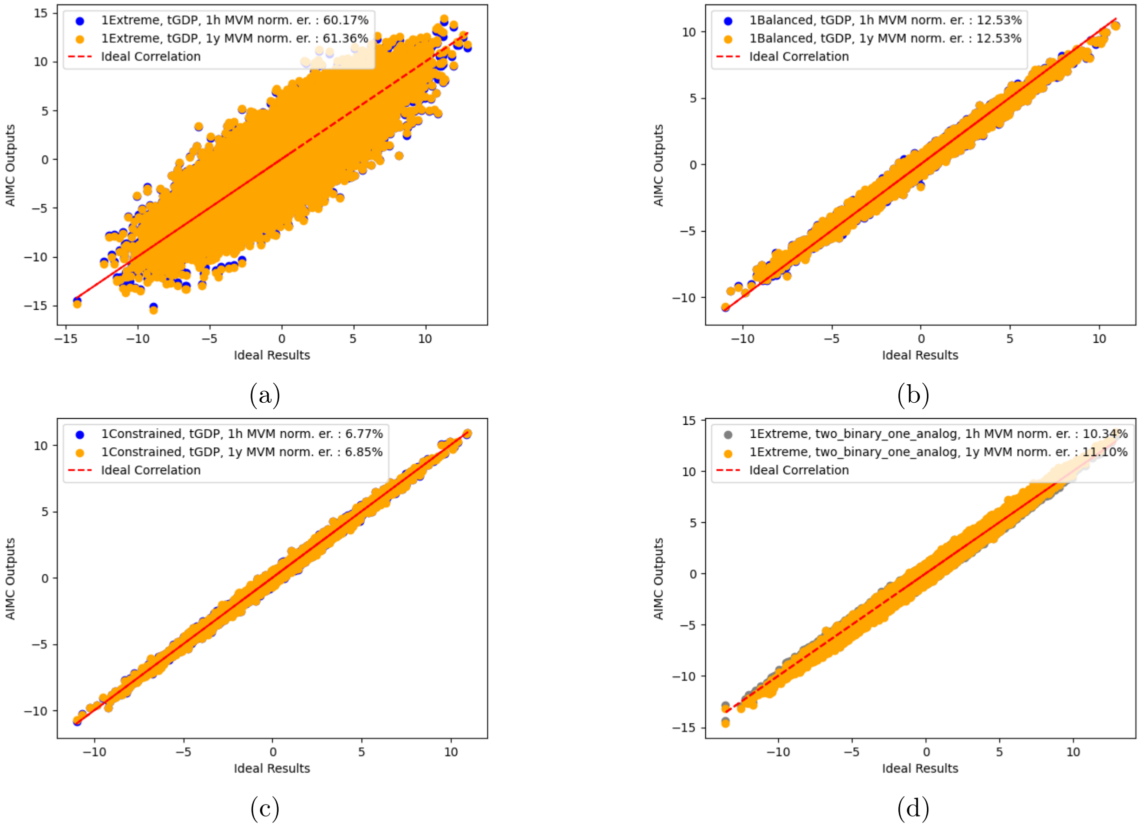
To evaluate the performance of an Analog In-Memory Computing (IMC) standard Vector

Matrix Multiplication (VMM) model, a same approach as presented in [43] was adopted. The VMM error was computed as the ratio of the l2-norm of the discrepancy between the actual and expected outcomes $(y - y_e)$ to the l2-norm of the ideal result (y). A simulation was conducted on a 512x512 array, using the peripheral circuit baseline depicted in Table 3.1. Weights were assigned using various programming methods. The VMMs were executed with Gaussian random weight matrices and uniformly random inputs.

The correlation between the AIMC output and the ideal output is demonstrated in Figures 4.3 and 4.4. In each graph, the weights underwent drifting for different durations, depicted as orange and blue for one hour and one year, respectively.

The initial four figures illustrate the disparity between the FTJ model and the programming method. The presented graphs indicate that the utilization of tGDP programming on the "Extreme" device leads to detrimental errors, thereby underscoring the imperative for an alternative and refined programming approach.

On the other hand, the last two figures exhibit the impact of increasing the number of devices per crosspoint in the case of tGDP weight programming for the two most promising FTJ models, namely "Balanced" and "Constrained.



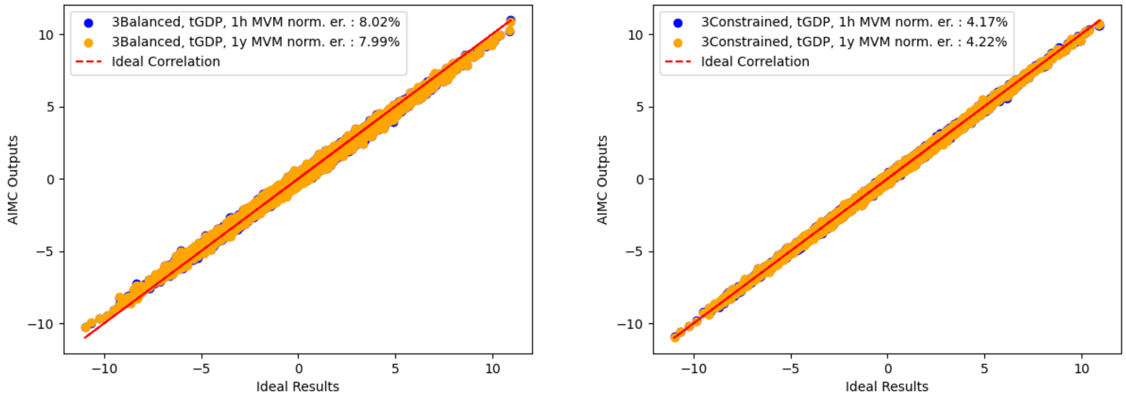
(a)

(b)

(c)

(d)

Figure 4.3: The graph displays the correlation of AIMC output with the Ideal one. The legend indicates the number of synapses per weight, the device model, the programming method, the drift time and the MVM normalized error. a) "Extreme" with tGDP b) "Balanced" with tGDP, c) "Constrained" with tGDP, d) "Extreme" with two binary one analog scheme



(a) (b)

Figure 4.4: The graph displays the correlation of AIMC output with the Ideal one. The legend indicates the number of synapses per weight, the device model, the programming method, the drift time and the VMM normalized error. a) 3 "Balanced" devices per crosspoint, b) 3 "Constrained" devices per crosspoint

4.2.2 Neural Network Inference

HWA training was introduced in the Methods section 3 and the findings are particularly prominent when comparing Tables 4.1, 4.2 and looking at Figure 4.5.

Figure 4.5 provides a visual comparison of the inference results obtained from the ResNet32 model when classifying the CIFAR-10 dataset. Each pair of bars in the graph represents a specific weight programming method. The blue bar corresponds to the accuracy achieved when the FP32 model is directly implemented on the analog hardware. On the other hand, the orange bar represents the accuracy when the HWA model is employed on the same hardware. The y-axis displays the normalized accuracy, denoted as A_{1h}, in percentage terms.

This graph underscores the crucial role that an appropriate programming method plays in achieving accurate results. The PCM preset, based on the [53] study, yields the highest accuracy for the HWA model. IBM has conducted extensive research into the weight programming technique for PCM, a level of analysis which is currently underway for the FTJ device. Despite this, the FTJ demonstrates promising potential, particularly with the "Constrained" device, showing results that are in the same level. Furthermore, it is evident that HWA training had greater impact on some programming techniques. This likely stems from the fact that during training the "tGDP Balanced" technique's effective error was accounted for. As a result, the algorithm adapted to this specific type of errors.

Tables 4.1 and 4.2 display the accuracy measurements of Neural Networks (NNs) utilizing the baseline AIMC paired with tGDP programming. These measurements were taken on a "Balanced" device, a selection made due to its feasibility for fabrication, under the conditions of one-hour and one-year conductance drift.

Table 4.1 provides the results of inference performed by directly deploying the model onto the analog hardware. In contrast, Table 4.2 presents the results after applying HWA training prior to the deployment onto the analog hardware.

These results represent the average accuracy from ten inference runs. Small grey numbers provide the standard deviation values for these averages.

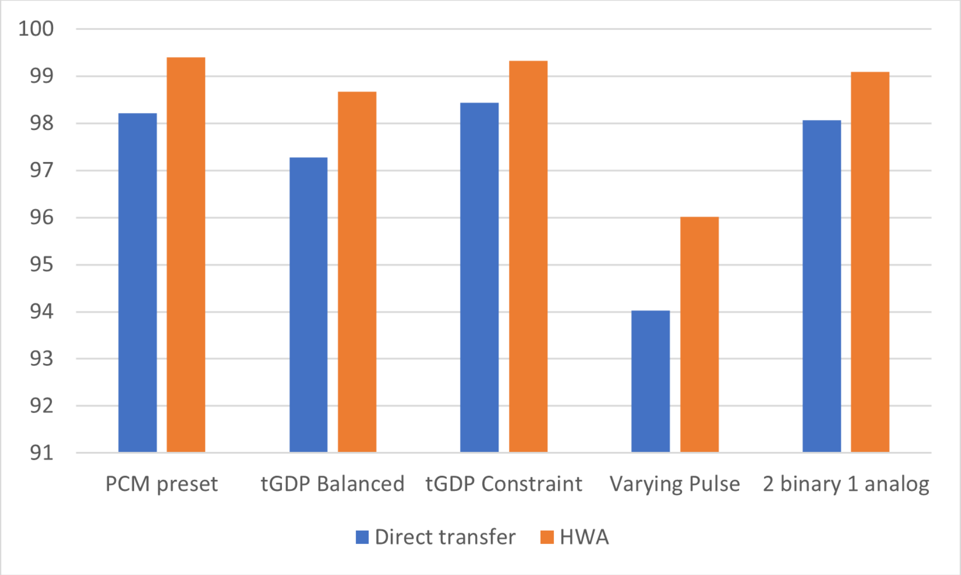


Figure 4.5: Inference accuracy of the ResNet32 model, deployed on AIMC, for classifying the CIFAR10 dataset. The horizontal signifies the programming technique employed, while the vertical represents the average normalized accuracy (A_{1h}) derived from ten inferences impacted by a one-hour drift. The blue bars denote the accuracy attained when the NN model is directly deployed on the analog hardware. Conversely, the orange bars reflect the accuracy of the NN model deployed after HWA training.

Table 4.1: The test error, represented as a percentage ± standard error of the mean (grey numbers), is reported for DNN deployment on AIMC crossbars after direct mapping. The test error is calculated across 10 inference repeats after reprogramming. Additionally, the rightmost two columns display the normalized accuracy at 1 hour and 1 year after weight programming. The normalized accuracy is scaled between the FP reference and chance error.

Direct mapping	Test Error (%)			Normalized Accuracy (%)	
Model	FP32	1 hour	1 year	A_*^{1h}	A_*^{1y}
ResNet-32	8.25	10.596 1.37	11.13 1.18	97.27	97.07
BERT - base	11.79	18.88 0.18	19.01 0.2	91.97	91.81
YoloV8-nano	50.48	72.61 0.87	74.68 0.62	55.34	51.14

Table 4.2: The test error, represented as a percentage ± standard error of the mean(grey numbers), is reported for DNN deployment on AIMC crossbars after HWA training. The test error is calculated across 10 inference repeats after reprogramming. Additionally, the rightmost two columns display the normalized accuracy at 1 hour and 1 year after weight programming. The normalized accuracy is scaled between the FP reference and chance error.

HWA training	Test Error (%)			Normalized Accuracy (%)	
Model	FP32	1 hour	1 year	A_*^{1h}	A_*^{1y}
ResNet-32	8.25	9.34 0.39	9.97 0.61	98.67	97.9
BERT - base	11.79	12.77 0.04	12.86 0.12	98.9	98.79
YoloV8-nano	50.48	58.85 0.51	61.33 0.43	83.1	78.1

The accuracy of YOLOv8-nano for segmentation may be considered low but, depending on the desired task, it can be usable. In Figure 4.6, despite some false predictions (e.g., table) and missing predictions (e.g., handbag), the detection of people is accurate. On the other hand, the direct mapping is not suitable in its current state as it can be seen in Figure 4.6. It fails to properly define people, utilizes multiple boxes to represent a single person, and produces incorrect predictions.

It should be noted that YOLOv8-nano is the smallest model within the YOLOv8 family and, consequently, exhibits certain imprecisions. The hardware-aware retraining was limited by the available hardware resources, with a maximum batch size of 8, even with the dual RTX3090 GPUs. There is potential for improvement in the retraining process, which might get closer to the ISO accuracy.

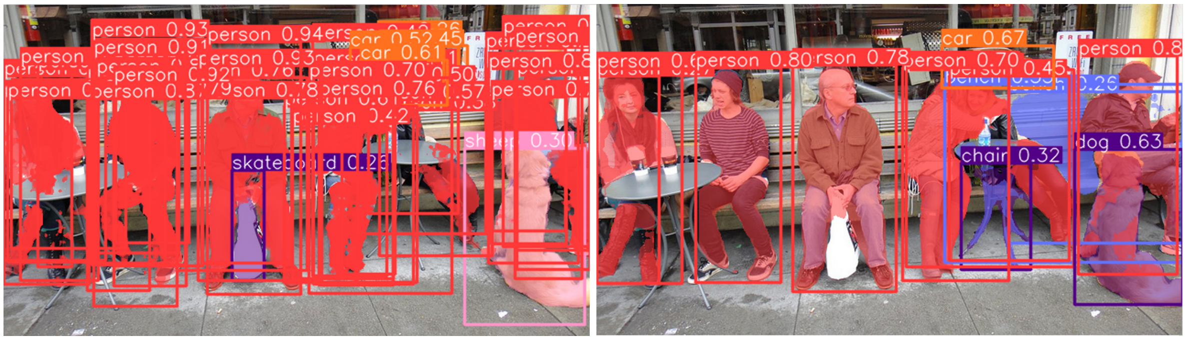


Figure 4.6: Example segmentation results after 1 hour drift with YOLOv8-nano. The left image is the prediction of direct mapping. The right image is the prediction after hardware-aware retraining.

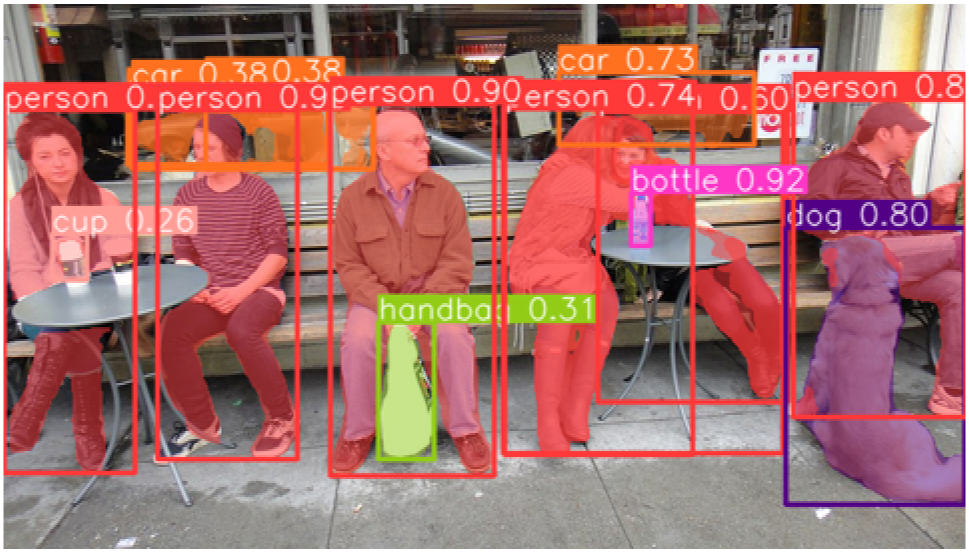


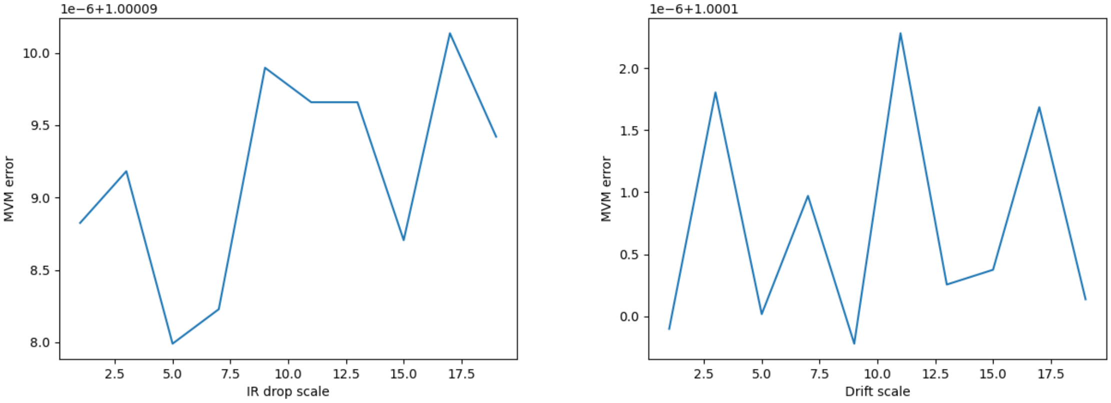
Figure 4.7: Segmentation prediction of the YOLOv8-nano digital model for reference.

4.2.3 Sensitivity Analysis

Through the implementation of hardware-aware (HWA) models, a set of experiments were conducted to assess the models' sensitivity to increasingly noisy hardware, as well as sensitivity on a per-layer basis. This analysis provides insights into potential hardware simplifications and identifies the layers that are tolerant to such modifications. Essentially, it becomes possible to segment the network into sections that require high precision and those that don't, thereby maintaining iso-accuracy.

Peripheral Circuitry

Starting with a simple 512 by 512 array, the effect of IR drop and drift due to FTJ devices is tested. In Figure 4.8a, it is clear the IR drop is negligible for the FTJ device. This is expected since as it was mentioned before the current through these devices is extremely small. As for the drift, Figure 4.8b shows that even if the drift increases as much the VMM error will not increase by a lot. The drift compensation is implemented here, but even without it the drift does not produce noticeable diversions.

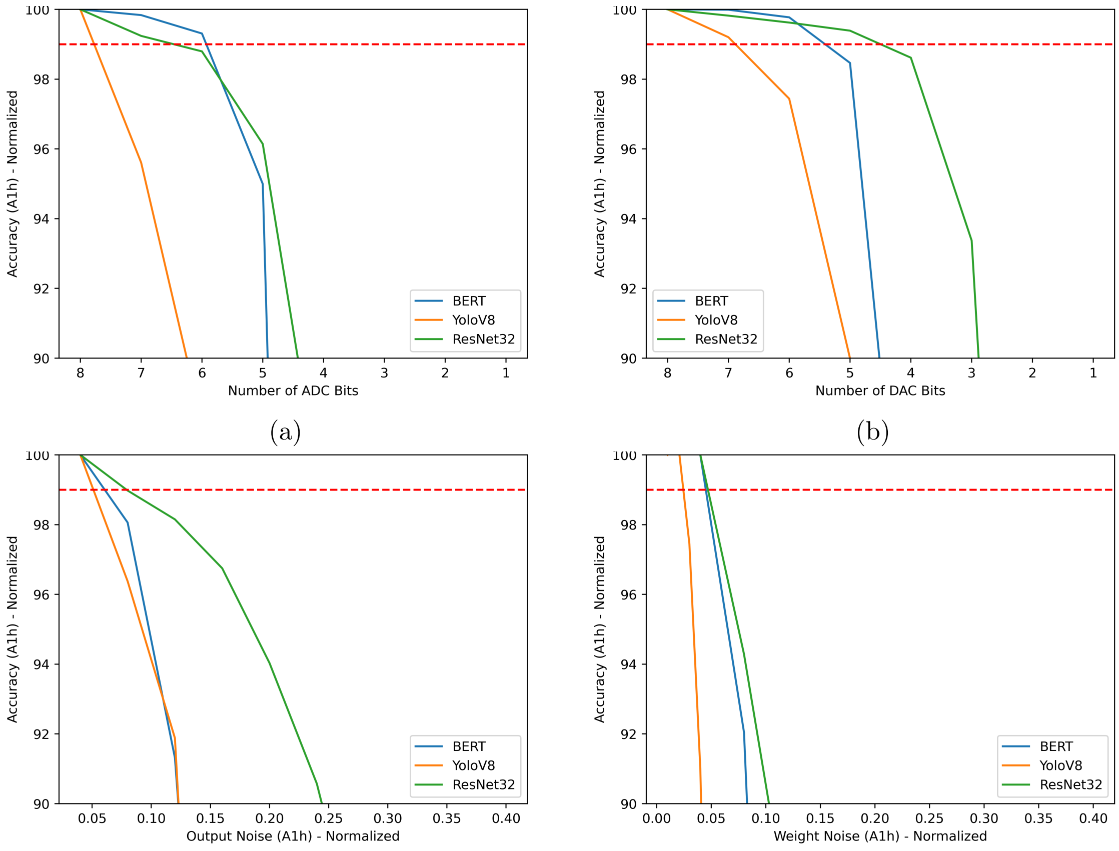


(a)

(b)

Figure 4.8: IR and drift effect on VMM error: a) IR drop, b) Drift

Since these two non-idealities are not very important, the next tests are focusing on the more sensitive parameters of ADC/DAC precision, system output referred noise and weight noise. In Figure 4.9, the sensitivity of the HWA retrained ResNet-32, YoloV8-nano and BERT-base on the introduction of pessimistic harware configurations are tested.



(a)

(b)

(c)

(d)

Figure 4.9: Responses of four AIMC imperfections across the tested ANNs. When just one error type is increased from baseline settings, the accuracy gradually drops to 99% (this is compared to the accuracy of the baseline AIMC model).(a) ADC , (b) DAC, (c) Ouput referred noise, (d) Weight noise.

Per layer

A layer-by-layer sensitivity analysis was conducted for both the BERT and Yolov8 models. To identify which layer was most susceptible, we individually transferred each layer to a tile with an output noise setting of 1 and ran an inference for each scenario. The top five most sensitive layers for both networks are displayed in Table 4.3.

Table 4.3: Top 5 most sensitive layers of BERT-base and Yolov8-nano, starting with most sensitive.

	BERT-base	Yolov8-nano
1	qa_outputs	SegmentationModel.model[4].cv2.conv
2	encoder.layer.4.intermediate.dense	SegmentationModel.model[12].m[0].cv2.conv
3	encoder.layer.2.intermediate.dense	SegmentationModel.model[3].conv
4	encoder.layer.5.intermediate.dense	SegmentationModel.model[0].conv
5	encoder.layer.1.intermediate.dense	SegmentationModel.model[1].conv

Equipped with this understanding, we organized the layers from most to least sensitive and carried out an experiment. This involved using the HWA models and progressively shifting each layer to a tile that was free from output and weight noise and featured flawless programming.

Figure 4.10 exhibits the resulting A_{1h} accuracy following each layer's transfer to the noise-free hardware. The x-axis represents the proportion of layers transferred, while the y-axis depicts the A_{1h} accuracy of the model at that stage. The horizontal red dashed line indicates the 99% mark of ISO-accuracy.

Upon analyzing the Figure 4.10, it is observed that Yolov8 could not attain 99% accuracy even when all layers were rendered noise-exempt. This is attributed to the drop in the accuracy of the HWA model compared to the FP32, despite it becoming more sensitive to non-idealities.

In contrast, the accuracy of the BERT model never achieves 100% due to the randomness of the noise, causing varying results that sometimes yield higher or lower values. The inference results presented are the average of five iterations, which, despite the averaging process, still does not result in a perfectly uniform line

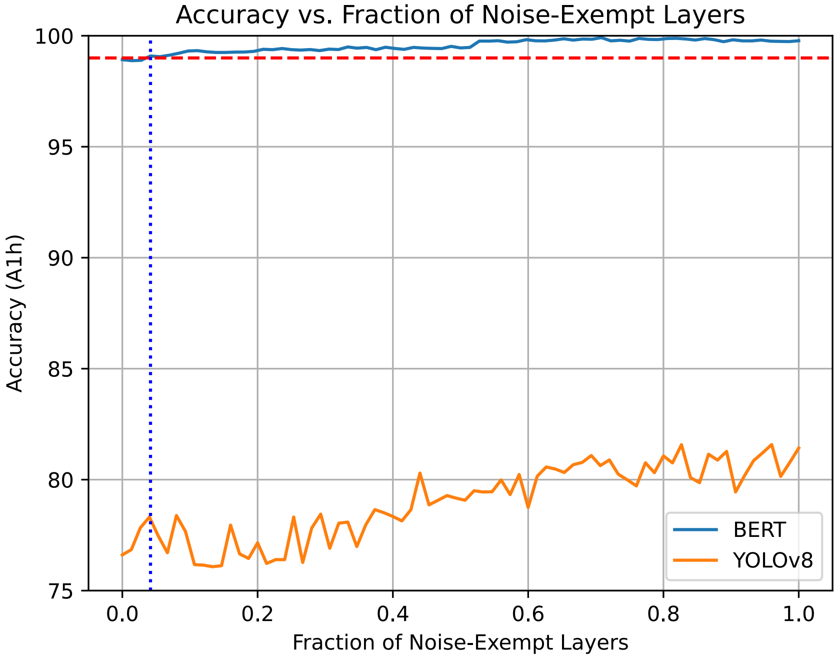


Figure 4.10: The accuracy (A_{1h}) of HWA BERT-base and Yolov8-nano when the most critical layers are excluded from FTJ noise. The plot illustrates how the accuracy changes as the fraction of noise-exempt layers increases, arranged in order from the most sensitive to the least sensitive.

The BERT model achieved over 99% normalized accuracy, thus reaching ISO-accuracy levels, by merely converting four layers to optimistic hardware. This encouraging outcome suggests a potential application scenario for AIMC technology. In contrast, Yolov8 showcases the importance of precise calculations on CNNs, especially on complex tasks, as is segmentation in big datasets as COCO. This results also align with the recent research from IBM [43].

Chapter 5

Discussion

In this chapter, we will discuss the limitation and possible improvement for training with FJTs. It will be followed by the limitation and possible improvement for inference with FTJs. Then some other considerations will be addressed.

By discussing the limitations, possible improvements, and other important considerations for AIMC, this chapter aims to provide a comprehensive understanding of the current challenges and opportunities using FTJs for AIMC.

5.1 AIMC for neural network training

5.1.1 Multi-layer Perceptron

As can be observed in Figure 4.1, the performance of the "Extreme" model of the FTJ device, characterized by the highest Δw_{min}, is unstable. The model fails to converge, exhibiting fluctuating behavior throughout the training. This instability can be attributed to the limitations of the update algorithm, which struggles to make minimal adjustments to the weights, thereby preventing steady convergence.

Despite the non-idealities of the array, the Tiki-Taka algorithm demonstrates an acceptable level of competence. It is worth noting that the algorithm's performance could potentially be further optimized with additional fine-tuning of the hyperparameters.

The "Balanced" device, which also exhibits an asymmetrical response but with a smaller Δw_{min}(=0.1), produces the best results when used in conjunction with the Tiki-Taka algorithm.

However, iso-accuracy is achieved with the "Constrained" device, which boasts a 50 times smaller Δw_{min} and a linear response. Despite its higher cycle-to-cycle variation, the device linearity and number of states show more importance. Training of the multi-layer Perceptron appears to be an easy fit with the proper RPU configuration.

5.1.2 ResNet-32

The findings indicate that even sophisticated configurations fall short of achieving training accuracy equivalent to FP-32. A configuration of 4 Constrained FTJ devices per crosspoint with ideal I/O circuitry could barely achieve 40% accuracy after 110 epochs of training.

Nonetheless, it's crucial to acknowledge that ResNet-32 is a complex neural network to train, even within a digital hardware. Identifying optimal hyperparameters presented a significant challenge, and it's worth noting that the state of the art results surpass those obtained using the selected hyperparameters.

Gokmen's study [34] demonstrated that Tiki-Taka version 2 requires a minimum of 15 states to achieve satisfactory performance during training. However, the trained neural network utilized in the study was an LSTM, a type of recurrent NN. It is essential to conduct similar experiments with the ResNet architecture as well. Unfortunately, the FTJ models do not meet the criteria of having more than 15 states at the time of this study, which prevented further exploration of hypothetical models in this work. As the technology improves this algorithm could prove beneficial and probably with proper NN architectures the training on AIMC will be feasible.

5.2 AIMC for inference

In terms of model's inference, AIMC demonstrates promising performance even in the presence of a 12.5% VMM error. The Neural Networks are able to adapt effectively despite the error. Moreover, in certain tasks, it is possible to disregard the noise altogether. This approach shares similarities with digital logic, where hardware can differentiate between a bit representing 1 or 0 despite the presence of noise.

5.2.1 VMM for inference

From the comparative analysis of the results, it is apparent that an effective programming method significantly improves the VMM error. The main take-aways of the VMM experiments are:

- The programming method can completely change the response of the model FTJ and as a result even the Extreme device can be functional. The varying pulse scheme should be further explored and implemented in a prototype array. It is the proper way of taking advantage of the whole on/off ratio of a FTJ memristor and finding an efficient method for applying it on hardware could unlock the complete potential of the device.
- The tGDP algorithm is an interesting approach that sets the weight matrix to a crossbar array in a way that tries to mitigate its own imperfections. However, the device should have a small response towards the pulses and be as linear as it can.
- Using multiple devices per crosspoint for the tGDP algorithm improves the result. Also there is potential configurations like weighted devices and deterministic pulsing generation that should be explored further.
- Finally, the binary mode can be integrated with the multi-level mode—employing a mixed-signal synaptic slicing technique—to enhance weight precision, thereby mitigating the influence of device non-idealities.

5.2.2 Neural Networks

In the case of ResNet-32, an impressive accuracy of 99.33% was achieved when utilizing the tGDP programming method on the constrained FTJ model 4.5. Comparing with the PCM preset provided by IBM simulator, which has extensively been studied and improved, the FTJ holds its ground. The programming technique has a vital role in the accuracy of a model. One astonishing result, is that the "Extreme" device when used with varying amplitude pulsing scheme can be viable option.

It should also be noted that, the HWA training was targeted for the tGDP programming method. If it was targeted for the "two binary one analog" method, there might have achieved better results for that. It is definitely something worth investigating.

Looking to the Tables 4.1, 4.2, it is apparent that HWA training before deploying the network on AIMC is extremely important. The approach of applying as similar to the hardware noise during the forward pass of the training is what makes the difference. The network tries to minimize the loss function and addresses the noise as a regularization technique, which in the end provides better generalization to the network.

The ResNet and BERT models achieve accuracy levels comparable to ISO standards, suggesting potential use cases. Particularly noteworthy is the performance of BERT, a network with an impressive architecture consisting of 109,096,136 parameters and 73 fully connected layers, making it a remarkable achievement for the AIMC model. When comparing it to Yolov8, a model with fewer layers, only 3.2 million parameters and 76 convolution layers, it becomes evident that the outcome is sensitive to both the layer structure and the nature of the task itself. In the case of Yolov8, the convolution kernel needs to be applied multiple times, and in the context of AIMC, the kernel cannot remain the same, resulting in divergence in the feature space. Additionally, the task of segmenting 640x640 pixels images is highly complex, and the evaluation metric is equally challenging.

5.3 Other considerations

5.3.1 Attention to Hardware Design

Figure 4.9 demonstrates the potential for optimizing energy efficiency. It appears that the precision of the ADC plays a crucial role, while different architectures exhibit the ability to handle lower resolutions. This presents an opportunity to employ adjustable tile architectures, where the precision of ADC and DAC can be reduced based on the specific network, resulting in even more efficient inference operations.

Another way of increasing the power efficiency and reducing the area of the AIMC chip is to use the ADCs to perform the activation functions. The digital approach uses the ADC to convert results and input them into a digital block that performs the activation. The paper [54] proposes to perform the activation functions during the conversion by using a combination of shared mapping function and columns with linear approximations. This approach removes the need of a separate activation circuit and allows to perform the activation at the same time as the MAC operation.

The results presented in Figure 4.9c,d provide insights into the significance of each imperfection. It is evident that both output and weight noise should be minimized as much as possible. While output noise is somewhat mitigated through noise management and weight scale mitigation techniques, addressing weight noise requires approaches such as averaging inference results, HWA training, or even altering the neural network architecture altogether. As we have seen, the topology of the neural network plays an important role on its sensitivity. The BERT model, which primarily consists of Fully Connected (FC) layers, demonstrates a comparable level of resistance to weight noise as the smaller ResNet-32 network. We assume that this occurrence can be attributed to two factors.

Firstly, the convolution operation relies on applying the same kernel multiple times. Therefore, any deviation in a weight within the crossbar array will disrupt this operation, resulting in an outcome that significantly diverges from the intended convolution.

Secondly, the specific task assigned to the algorithm matters. For example, tasks such as object classification or predicting words for question-answering do not require the same level of precision as object segmentation within an image. In less sensitive cases like the former, the network computes a probability, and the exact value doesn't significantly influence the final decision. On the other hand, tasks like segmentation evaluate the result for each pixel, where even a single misclassified pixel can lead to a reduction in overall precision.

Before deploying a network, it is crucial to conduct a thorough sensitivity analysis. This examination should assess how each layer of the network is affected by a decrease in hardware precision. As displayed in Table 4.3, specific layers display substantial sensitivity. For instance, in the Bert model, layers such as 'dense' (5.1) —which possess the highest number of parameters—and 'qa_output'—which compute the probabilities of the outcome—are highly sensitive. In contrast, in the Yolov8 model, layers with considerable sensitivity are predominantly located in the first layers.

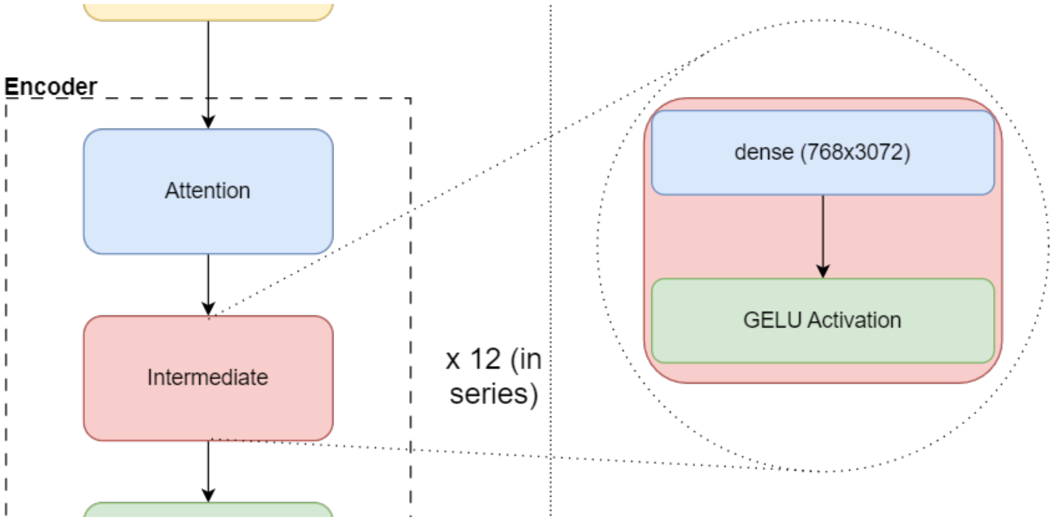


Figure 5.1: BERT's dense layers

Regarding the dense layers, they are of 768x3072 size and our hypothesis is that their increased sensitivity could be a function of their higher parameter count compared to the rest. However, this hypothesis necessitates further exploration for validation. In relation to the 'qa_output' layer, it possesses dimensions of 768x2 but this layer's role is to determine whether the currently processed word signifies the start or/and the end of an answer. If the data in this layer is subject to noise, it could mistakenly deduce that a given word represents both the beginning and the end of the answer. This could potentially result in a misprediction.

For the Yolov8, the layers nearer to the beginning also possess the highest dimensionality, meaning that the kernel is subjected to a higher number of convolutions. It also means that the errors at the first stages will propagate to the rest of the NN. As previously mentioned, this is a complex process for AIMC, given the imperfect programming of weights. Furthermore, layers with a 1x1 kernel present distinct challenges, as the same weight requires exact programming across the entire crossbar array.

Interestingly, even a slight relocation of these layers to an optimized hardware configuration can enable the BERT-base network to achieve ISO accuracy levels. By incorporating this analysis, layers can be assigned to specific Tiles based on their sensitivity, thereby optimizing energy efficiency while maintaining accurate inference results.

5.3.2 Research Ideas

The AIMC domain continues to offer a wealth of possibilities for investigation and experimentation with various design considerations. One area of particular interest is the development of neural network structures that may demonstrate greater resilience to AIMC imperfections. A noteworthy suggestion in this regard is to contemplate expanding the width of these networks rather than their depth, given that noise tends to amplify as it journeys through deeper layers.

Additionally, as previously discussed, there's potential to execute activation functions using the ADC. This approach not only eliminates the requirement for data to traverse digital logic but could also enhance the ADC's precision by one bit, given the opportunity to dismiss negative values. It would be beneficial to validate this hypothesis using tools akin to SPICE, providing a more comprehensive perspective.

Moreover, the configuration of the RPU in tandem with the weight programming method warrants further scrutiny. Experimental findings have illustrated that weight programming plays a critical role in the VMM error. It is worth noting that the RPU's potential is not confined to the use of a memristor alone. Combining transistors with the memristor or even incorporating configurations akin to a Wheatstone bridge [55] present intriguing concepts worthy of exploration.

Other applications might also benefit from the AIMC where precise computation isn't mandatory. Some examples could be filtering, image processing or in system control where the noise introduced by the loss of accuracy is acceptable or could be compensated.

Nevertheless, all of these ideas need a capable simulator that closely matches the real-world conditions. The IBM AIHWkit is still under development and it needs more people to help it grow. It should be noted however that capable graphic processing units are needed for this area of research. And it was the main challenge for this work as well.

Chapter 6

Conclusion

In essence, the objective of this research was to assess the application of Ferroelectric tunnel junction (FTJ) as a resistive processing unit for analog in-memory computing, particularly for advanced deep neural networks. The investigation included both neural network training and inference, with a stronger emphasis on the latter. Four architectures were used, namely a multilayer-perceptron with three fully connected layers, a modified ResNet architecture, the BERT-base transformer for natural language processing, and YOLOv8-nano for image segmentation tasks. While the first two networks were evaluated for training purposes, ResNet was also tested for inference, although the primary focus was on BERT and YOLOv8 due to their potential real-world applications in the AIMC paradigm. Additionally, a sensitivity analysis was conducted to examine the impact of various sources of device and peripheral circuit noise on the accuracy of the models. Throughout the experiments, approaches were explored to mitigate the effects of these non-idealities.

The work proved the FTJ to be capable memristor for inference task. Furthermore, the transformers are more suitable for AIMC compared to the conventional CNNs which are more sensitive to non-idealities and harder to retrain. It was possible to reach the ISO accuracy with BERT while YOLO v8 reached a maximum accuracy of 83% compared to the digital model. Retraining in a hardware-aware manner helped improve the accuracy of different DNN models which confirm the utility of this procedure for deploying DNNs in analog chips. Concerning the peripheral circuitry, using 8-bits ADCs and 7-bits DACs are sufficient to reach acceptable accuracy. The models are sensitive to output and weight noises, they only accept few percent of noise ratio ($\approx 5\%$) to maintain proper accuracy. Which means these parameters must be carefully scrutinized for the future implementations. An essential criteria to obtain proper accuracy is the ability to set the conductances precisely. To do so a varying amplitude pulse approach and a multi-RPU architecture are proposed.

However this study also shows that training in AIMC is a demanding task, and the current FTJ models does not have enough states to encode the needed information for this task. Nevertheless, intensive work is ongoing to improve the memristors. During the course of the project, it has been shown the FTJs could now have more than 60 reliable states.

One direction for future research is to focus on FTJ programming to achieve higher states that will help the programming and eventually enable the possibility of training. Evaluating the AIMC on different tasks than AI might be interesting. The embedded world could benefit from this efficient and compact computation in many ways even with it's limited precision.

Bibliography

[1] Copeland, B.J., 2000. The modern history of computing.

[2] "The Nobel Prize in Physics 1956". Nobelprize.org. Nobel Media AB. Archived from the original on December 16, 2014. Retrieved December 7, 2014.

[3] Balasubramanian, V., 2021. Brain power. Proceedings of the National Academy of Sciences, 118(32), p.e2107022118. doi: 10.1073/pnas.210702211

[4] Chua, L., 1971. Memristor-the missing circuit element. IEEE Transactions on circuit theory, 18(5), pp.507-519.

[5] Strukov, D.B., Snider, G.S., Stewart, D.R. and Williams, R.S., 2008. The missing memristor found. nature, 453(7191), pp.80-83.

[6] Rasch, M.J., Moreda, D., Gokmen, T., Le Gallo, M., Carta, F., Goldberg, C., El Maghraoui, K., Sebastian, A. and Narayanan, V., 2021, June. A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays. In 2021 IEEE 3rd international conference on artificial intelligence circuits and systems (AICAS) (pp. 1-4). IEEE. doi: 10.1109/AICAS51828.2021.9458494.

[7] Li, Z., Liu, F., Yang, W., Peng, S. and Zhou, J., 2021. A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems.

[8] Fukushima, K., 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological cybernetics, 36(4), pp.193-202. doi:10.1007/BF00344251

[9] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2017. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), pp.84-90.

[10] Indolia, S., Goswami, A.K., Mishra, S.P. and Asopa, P., 2018. Conceptual understanding of convolutional neural network-a deep learning approach. Procedia computer science, 132, pp.679-688.

[11] McCelloch, W.S. and Pitts, W., 1943. A logical calculus of the idea immanent in neural nets. Bulletin ofMathematical Biophysics, 5, pp.115-133. doi:10.1007/BF02478259

[12] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems, 30.

[13] Prodromakis, T. and Toumazou, C., 2010, December. A review on memristive devices and applications. In 2010 17th IEEE international conference on electronics, circuits and systems (pp. 934-937). IEEE. doi: 10.1109/ICECS.2010.5724666.

[14] Athle, R. and Borg, M., 2023. Impact of Temperature-Induced Oxide Defects on $Hf_xZr_{1x}O_2$ Ferroelectric Tunnel Junction Memristor Performance. IEEE Transactions on Electron Devices, 70(3), pp.1412-1416.

[15] Valasek, J., 1921. Piezo-electric and allied phenomena in Rochelle salt. Physical review, 17(4), p.475.

[16] Böscke, T.S., Müller, J., Bräuhaus, D., Schröder, U. and Böttger, U., 2011. Ferroelectricity in hafnium oxide thin films. Applied Physics Letters, 99(10), p.102903. doi.org/10.1063/1.3634052

[17] Pantel, D. and Alexe, M., 2010. Electroresistance effects in ferroelectric tunnel barriers. Physical Review B, 82(13), p.134105.

[18] Örnhag, M.V., Güler, P., Knyaginin, D. and Borg, M., 2023. Accelerating AI Using Next-Generation Hardware: Possibilities and Challenges With Analog In-Memory Computing. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 488-496).

[19] Saul J, Bass D. Artificial Intelligence Is Booming—So Is Its Carbon Footprint. Bloomberg.com. March 2023:N.PAG.

[20] Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J.P., Hu, M., Williams, R.S. and Srikumar, V., 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. ACM SIGARCH Computer Architecture News, 44(3), pp.14-26.

[21] Bojnordi, M.N. and Ipek, E., 2016, March. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 1-13). IEEE.

[22] Xiao, T.P., Bennett, C.H., Feinberg, B., Agarwal, S. and Marinella, M.J., 2020. Analog architectures for neural network acceleration based on non-volatile memory. Applied Physics Reviews, 7(3). doi.org/10.1063/1.5143815

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

[24] Huang, H., Du, L. and Chiu, Y., 2017. A 1.2-GS/s 8-bit two-step SAR ADC in 65-nm CMOS with passive residue transfer. IEEE Journal of Solid-State Circuits, 52(6), pp.1551-1562. doi: 10.1109/JSSC.2017.2682839.

[25] Kull, L., Toifl, T., Schmatz, M., Francese, P.A., Menolfi, C., Braendli, M., Kossel, M., Morf, T., Andersen, T.M. and Leblebici, Y., 2013. A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital SOI CMOS. IEEE Journal of Solid-State Circuits, 48(12), pp.3049-3058. doi: 10.1109/JSSC.2013.2279571.

[26] Ohhata, K., 2019. A 2.3-mW, 1-GHz, 8-bit fully time-based two-step ADC using a high-linearity dynamic VTC. IEEE Journal of Solid-State Circuits, 54(7), pp.2038-2048. doi: 10.1109/JSSC.2019.2907401.

[27] Oh, D.R., Moon, K.J., Lim, W.M., Kim, Y.D., An, E.J. and Ryu, S.T., 2020. An 8-bit 1-GS/s asynchronous loop-unrolled SAR-flash ADC with complementary dynamic

amplifiers in 28-nm CMOS. IEEE Journal of Solid-State Circuits, 56(4), pp.1216-1226. doi: 10.1109/JSSC.2020.3044624.

[28] Chen, H., Zhot, X., Yu, Q., Zhang, F. and Li, Q., 2018, June. A> 3ghz erbw 1.1 gs/s 8b two-sten sar adc with recursive-weight dac. In 2018 IEEE Symposium on VLSI Circuits (pp. 97-98). IEEE. doi: 10.1109/VLSIC.2018.8502370.

[29] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H. and Kalenichenko, D., 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

[30] Jain, S., Ankit, A., Chakraborty, I., Gokmen, T., Rasch, M., Haensch, W., Roy, K. and Raghunathan, A., 2019. Neural network accelerator design with resistive crossbars: Opportunities and challenges. IBM Journal of Research and Development, 63(6), pp.10-1.

[31] Rumelhart, D.E., Hinton, G.E. and Williams, R.J., 1986. Learning representations by back-propagating errors. nature, 323(6088), pp.533-536.

[32] Gaines, B.R., 1967, April. Stochastic computing. In Proceedings of the April 18-20, 1967, spring joint computer conference (pp. 149-156).

[33] Gokmen, T. and Haensch, W., 2020. Algorithm for training neural networks on resistive device arrays. Frontiers in neuroscience, 14, p.103.

[34] Gokmen, T., 2021. Enabling training of neural networks on noisy hardware. Frontiers in Artificial Intelligence, 4, p.699148.

[35] Rasch, M.J., Carta, F., Fagbohungbe, O. and Gokmen, T., 2023. Fast offset corrected in-memory training. arXiv preprint arXiv:2303.04721.

[36] Khaddam-Aljameh, R., Stanisavljevic, M., Mas, J.F., Karunaratne, G., Braendli, M., Liu, F., Singh, A., Müller, S.M., Egger, U., Petropoulos, A. and Antonakopoulos, T., 2021, June. HERMES Core–A 14nm CMOS and PCM-based In-Memory Compute Core using an array of 300ps/LSB Linearized CCO-based ADCs and local digital processing. In 2021 Symposium on VLSI Circuits (pp. 1-2). IEEE.

[37] Fuller, E.J., Gabaly, F.E., Léonard, F., Agarwal, S., Plimpton, S.J., Jacobs-Gedrim, R.B., James, C.D., Marinella, M.J. and Talin, A.A., 2017. Li-ion synaptic transistor for low power analog computing. Advanced Materials, 29(4), p.1604310.

[38] Kim, S., Todorov, T., Onen, M., Gokmen, T., Bishop, D., Solomon, P., Lee, K.T., Copel, M., Farmer, D.B., Ott, J.A. and Ando, T., 2019, December. Metal-oxide based, CMOS-compatible ECRAM for Deep Learning Accelerator. In 2019 IEEE International Electron Devices Meeting (IEDM) (pp. 35-7). IEEE. doi: 10.1109/IEDM19573.2019.8993463.

[39] Tang, J., Bishop, D., Kim, S., Copel, M., Gokmen, T., Todorov, T., Shin, S., Lee, K.T., Solomon, P., Chan, K. and Haensch, W., 2018, December. ECRAM as scalable synaptic cell for high-speed, low-power neuromorphic computing. In 2018 IEEE International Electron Devices Meeting (IEDM) (pp. 13-1). IEEE. doi: 10.1109/IEDM.2018.8614551.

[40] Rasch, M.J., Moreda, D., Gokmen, T., Le Gallo, M., Carta, F., Goldberg, C., El Maghraoui, K., Sebastian, A. and Narayanan, V., 2021, June. A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays. In

2021 IEEE 3rd international conference on artificial intelligence circuits and systems (AICAS) (pp. 1-4). IEEE. doi: 10.1109/AICAS51828.2021.9458494.

[41] Chen, P.Y., Peng, X. and Yu, S., 2017, December. NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures. In 2017 IEEE International Electron Devices Meeting (IEDM) (pp. 6-1). IEEE.

[42] Jain, S., Sengupta, A., Roy, K. and Raghunathan, A., 2020. RxNN: A framework for evaluating deep neural networks on resistive crossbars. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(2), pp.326-338.

[43] Rasch, M.J., Mackin, C., Gallo, M.L., Chen, A., Fasoli, A., Odermatt, F., Li, N., Nandakumar, S.R., Narayanan, P., Tsai, H. and Burr, G.W., 2023. Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators. arXiv preprint arXiv:2302.08469.

[44] Ott, H.W. (1998) 'Intrinsic Noise Sources', in Noise reduction techniques in electronic systems. Singapore: John Wiley, pp. 228–243.

[45] Deng, L., 2012. The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE signal processing magazine, 29(6), pp.141-142.

[46] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[47] Hope, T., Resheff, Y.S. and Lieder, I. (2017) 'CIFAR10', in Learning tensorflow: A guide to building deep learning systems. Beijing, China: O'Reilly, pp. 61–62.

[48] Everingham, M., Van Gool, L., Williams, C.K., Winn, J. and Zisserman, A., 2010. The pascal visual object classes (voc) challenge. International journal of computer vision, 88, pp.303-338. doi.org/10.1007/s11263-009-0275-4.

[49] Rajpurkar, P., Zhang, J., Lopyrev, K. and Liang, P., 2016. Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250.

[50] Sasaki, Y., 2007. The truth of the F-measure. 2007. URL: https://www. cs. odu. edu/mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07. pdf [accessed 2021-05-26], 49.

[51] Le Gallo, M., Krebs, D., Zipoli, F., Salinga, M. and Sebastian, A., 2018. Collective structural relaxation in phase-change memory devices. Advanced Electronic Materials, 4(9), p.1700627.

[52] Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A. and Boyle, R., 2017, June. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th annual international symposium on computer architecture (pp. 1-12).

[53] Joshi, V., Le Gallo, M., Haefeli, S., Boybat, I., Nandakumar, S.R., Piveteau, C., Dazzi, M., Rajendran, B., Sebastian, A. and Eleftheriou, E., 2020. Accurate deep neural network inference using computational phase-change memory. Nature communications, 11(1), p.2473.

[54] Giordano, M., Cristiano, G., Ishibashi, K., Ambrogio, S., Tsai, H., Burr, G.W. and Narayanan, P., 2019. Analog-to-digital conversion with reconfigurable function mapping for neural networks activation function acceleration. IEEE Journal on Emerging

and Selected Topics in Circuits and Systems, 9(2), pp.367-376. doi: 10.1109/JETCAS.2019.2911537.

[55] Wang, S., Song, L., Chen, W., Wang, G., Hao, E., Li, C., Hu, Y., Pan, Y., Nathan, A., Hu, G. and Gao, S., 2023. Memristor-Based Intelligent Human-Like Neural Computing. Advanced Electronic Materials, 9(1), p.2200877.

[56] Lin, T., Wang, Y., Liu, X., & Qiu, X. (2022). A survey of transformers. AI Open, Volume 3. https://doi.org/10.1016/j.aiopen.2022.10.001