Politecnico di Torino

IBM

Master's Thesis - IBM Research, Zurich

Claudia Maddalena - s315000

Impact of Hardware Non-Idealities on Analog In-Memory Neural Network Inference: Simulation and Validation

Politecnico di Torino

MSc. in Electronics Engineering - Microelectronics

Supervision

Politecnico di Torino:
Prof. Fernando Corinto
Prof. Alon Ascoli

IBM Research Zurich:
Victoria Clerico
Dr. Wooseok Choi
Dr. Valeria Bragaglia

A.Y. 2024/25

Acknowledgements

I would like to express my sincere gratitude to all those who supported me throughout the course of this thesis.

First, I would like to thank all the members of the Neuromorphic Devices and Systems Groups at IBM. Thanks to the group manager, Bert, and to each of you for welcoming me. From the very beginning, I felt fully included and appreciated, despite being just a thesis student. Your attention and respect truly meant to me. During these six months, I have learned a lot, both professionally and personally. It was an honor for me to join your group and I will always cherish the experience and the knowledge I gained. Being part of such a supportive and inspiring environment has been invaluable.

Many thanks to Valeria for giving me this incredible opportunity and for guiding me throughout this journey, and to my IBM supervisors, Mavi and Wooseok. Your ambition and dedication inspires me every day. Thank you for supporting me, listening to all my doubts and answering to all my questions. I have learned so much from you and I will forever be thankful to have met such wonderful and admirable people.

Furthermore, I would like to thank my internal supervisors, Prof. Fernando Corinto and Prof. Alon Ascoli, for the support from the very start, and for always being available and encouraging throughout my work.

A special thanks to all my friends, near and far. During these years, you listened to me patiently, and supported me through every ups and downs. Thank you for always bringing smiles and lightness to my days, and for being there at every milestone along the way.

Last but not least, a heartfelt thanks to Aldo and my family. You shared with me every moment of this journey, offering unconditional love and support day by day. Thank you for standing by my side through all the challenges, always reminding me how proud you are. I wouldn't be here without your constant encouragement.

Thank you all,

Claudia.

Abstract

In the last decade, the increasing complexity and energy demands of Artificial Intelligence (AI) applications have highlighted the limitations of traditional von Neumann computing architectures, where the physical separation between memory and processing units introduces a performance bottleneck from data passing. To overcome this challenge, Analog In-Memory Computing (AIMC) emerges as an energy-efficient alternative by enabling parallel in-memory Matrix-Vector Multiplication (MVM)-a core operation in AI models- significantly improving energy efficiency and latency by bypassing data transferring. AIMC cores accelerate MVMs exploiting a crossbar array architecture, which encodes the synaptic weights into cross point conductance levels. Among the emerging non-volatile memory technologies, Resistive Random Access Memory (ReRAM) devices provide multi-bit storage capabilities, suitable for accelerating neural network operations.

Novel ReRAM technology based on CMO-HfOx has further demonstrated enhanced properties, such as long-term state retention, low stochasticity, and high-precision memory programming and reading.

This work explores the potential of employing CMO-HfOx-ReRAM technology for neural network inference, by performing realistic hardware-aware simulations based on electrical characterization. This study investigates the impact of intrinsic device non-idealities on neural network inference over time, considering limitations at a crossbar array level, and subsequently broadening to different neural network architectures for performance evaluation. Across the different time-dependent defects of ReRAM devices, this work focuses on read noise, where a statistical model is derived and validated with experimental data. The read noise model complements existing programming noise and conductance relaxation models, providing a more comprehensive framework for inference simulation. These models are integrated into IBM Analog Hardware Acceleration Kit, an open-source simulation toolkit for performing hardware-informed on-chip inference and training simulations. The impact of intrinsic noise is analyzed on three

neural network architectures: MLP, LeNet-5, and ResNet-32 for different tasks. Inference accuracy is monitored over extended time periods to assess the robustness of each architecture. In this context, the benefits of Hardware-Aware Training (HWA) are also investigated as a strategy to mitigate accuracy degradation caused by hardware non-idealities.

Keywords: Neural Network Inference, Resistive RAM, Read noise, Conductance relaxation, IR drop, Hardware-Aware Training

Contents

Acronyms

A-IMC	Analog In-Memory Computing.
ADC	Analog to Digital Converter.
AI	Artificial intelligence.
AIHWKIT	IBM Analog Hardware Acceleration Kit.
ALU	Arithmetic Logic Unit.
ANN	Artificial Neural Network.
BL	Bit Line.
CBA	CrossBar Array.
CMO	Conductive Metal Oxide.
CMO-ReRAM	Conductive Metal Oxide/HfOx-ReRAM.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
DAC	Digital to Analog Converter.
DNN	Deep Neural Network.
DRAM	Dynamic Random access Memory.
FeRAM	Ferroelectric RAM.
HRS	High Resistance State.
HWA	Hardware-Aware.
IMC	In-Memory Computing.
LRS	Low Resistance State.

MAC Multiply-Accumulate.

MIM Metal-Insulator-Metal.

ML Machine Learning.

MLP Multi-Layer Perceptron.

MRAM Magnetic RAM.

MVM Matrix-Vector Multiplication.

NMC Near-Memory Computing.

NVM Non-Volatile Memory.

PCM Phase-Change Memory.

ReRAM Resistive Random Access Memory.

RMSE Root Mean Squared Error.

RTN Random Telegraph Noise.

SGD Stochastic Gradient Descent.

SL Source Line.

SRAM Static Random Access Memory.

TEL Thermal Enhanced Layer.

WL Word Line.

List of Figures

1 Theory

1.1 Neural Networks and Computational Demands

From search engines and personalized recommendations to voice assistants and autonomous driving, machine learning technologies play a central role in many contemporary digital services. Among these methods, deep learning, which uses Artificial Neural Network (ANN) with multiple layers, has demonstrated remarkable success in automatically extracting complex features from raw data, reducing the need for manual feature engineering [18].

Originally conceived in the 1940s to emulate the behavior of biological neurons, ANNs perform computations in a naturally parallel fashion. The introduction of modern graphical processing units (GPUs) has enabled the training of increasingly large networks on massive datasets, accelerating the development of deep neural networks (DNNs). These models now achieve near-human—or even superhuman—performance in diverse domains such as image recognition, speech understanding, language translation and gameplay [24]. As a result, deep learning systems have become widespread in commercial applications, powering social media platforms, recommendation engines, customer service automation, financial systems, and mobile technologies [24].

At the foundation of deep learning lies an idea inspired by the structure and functioning of the human brain. In fact, just as biological neurons in the brain process, each neuron in an ANN receives an input signal that is processed by the network and then the result flows out. A standard neural network usually consists of an input layer, one or more hidden layers, and an output layer (Figure 1.1). Each node connects to others, and has its own associated weights.

The most common applications of ANNs are classification and pattern recognition, prediction and modeling [1]. To perform these tasks effectively, neural networks rely on training data to learn meaningful patterns and progressively improve their accuracy. This begins with a forward inference step where the input is propagated through the network layer by layer to produce an output.

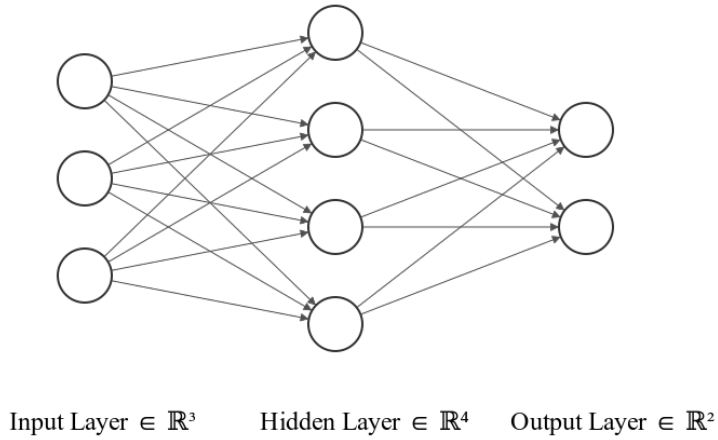


Figure 1.1: Neural Network basic scheme.

At each layer, the input activations are multiplied by a weight matrix, producing a new set of activations through a Matrix-Vector Multiplication (MVM). This operation can be expressed as:

$$\mathbf{y} = f(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}), \tag{1.1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ b is the bias vector, and $f(\cdot)$ is the activation function applied element-wise. This operation can be seen as a sequence of Multiply-Accumulate (MAC) steps. An activation function is applied after the linear combination of inputs and weights to introduce non-linearity into the model. This step is essential, as without it, the entire network would behave like a simple linear model, regardless of its depth [24]. The activation function maps the resulting values into a specific range, commonly between 0 and 1 (as in the sigmoid function) or - 1 and 1 (as in tanh), allowing the network to capture complex patterns in the data. In classification tasks, it helps the network make decisions by shaping the output toward interpretable values such as "yes" or "no" [20].

However, in the beginning, since the weights are initialized randomly, the output of the forward pass does not yet reflect the correct input–output mapping [24]. Training a neural network aims to progressively adjust these weights to minimize the difference between predicted and actual outputs. After each forward pass, the network evaluates its performance using a loss function, which quantifies the

error in its predictions [10]. Common loss functions include mean squared error for regression and cross-entropy loss for classification tasks.

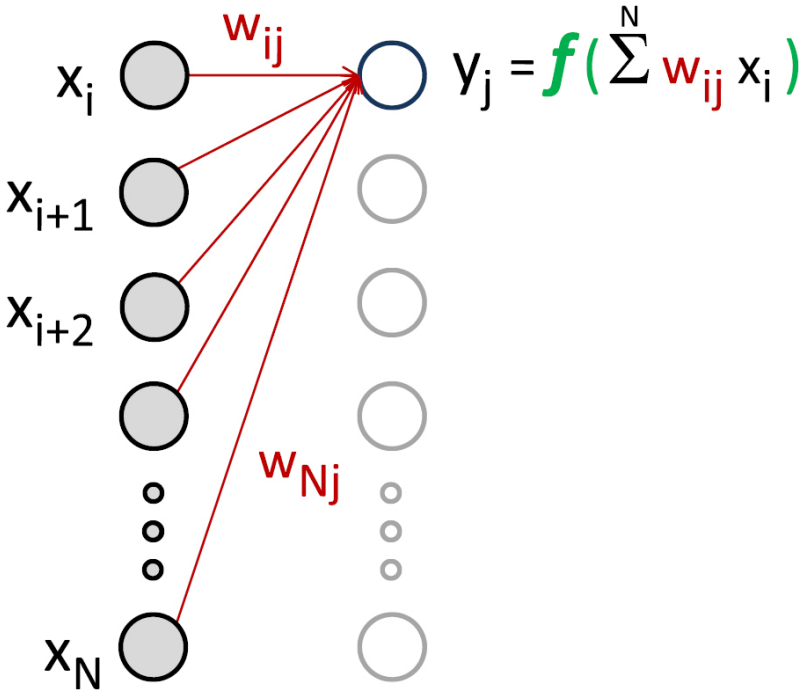


Figure 1.2: Illustration of a matrix–vector multiplication (MVM) in a neural network layer. [24]

To optimize performance, the backpropagation algorithm is used to compute the partial derivatives of the loss function with respect to all weights and biases, using the chain rule [10]. These gradients indicate how to update each parameter to reduce the overall error. The weights are then usually updated through an optimization procedure known as Stochastic Gradient Descent (SGD). Instead of computing the exact gradient over the entire training set, SGD estimates it using only a small, randomly selected batch of examples at each step. For each mini-batch, the model performs a forward pass, evaluates the prediction error, computes an approximate gradient, and updates the weights accordingly [18]. This process is repeated across many mini-batches until the loss function stops decreasing significantly. The stochastic nature of this approach introduces variability, i.e. "noise", in the gradient estimates, which not only speeds up training, but can also help the optimizer escape shallow or poor local minima. Despite its simplicity, SGD often leads to effective solutions faster than more complex optimization techniques [18]. During each update, weights are adjusted in the direction opposite to the

gradient, scaled by a learning rate that controls the magnitude of each update. This cycle of forward propagation, loss evaluation, backpropagation, and weight update repeats over many iterations until the loss converges to a minimum, improving the network's ability to generalize to new inputs [10].

After training, the system's performance is evaluated using a separate set of data known as the test set. This assessment, called *inference* checks how well the model can generalize, i.e., how accurately it can handle new, previously unseen inputs [18].

As deep learning models achieve increasingly high performance across tasks, they also grow in size and complexity. Consequently, the computational and energy costs required to train and deploy them have risen dramatically. Recent studies indicate that this growth is becoming unsustainable: while larger models tend to yield better results, the compute required scales disproportionately—often outpacing hardware improvements—thus emphasizing the need for more efficient computing architectures [23]. This growing demand highlights not only the limitations of current algorithmic approaches but also the physical and energy constraints of traditional CMOS-based digital systems. Despite the impressive gains in performance enabled by CMOS scaling over the past four decades—as predicted by Moore's and Dennard's laws—the pace of improvement has significantly slowed in recent years [14]. While early transistor miniaturization led to greater processing speeds and reduced power per operation, continued scaling has resulted in increased power density and thermal challenges, placing physical limits on further enhancements [14]. At the same time, the exponential growth of data generated by ubiquitous sensors and connected devices has imposed an unsustainable energy demand on conventional computing systems. Tasks like image recognition, real-time video analysis, and natural language processing demand vast computational resources, with training large neural models often consuming megawatt-hours of energy [13]. These challenges underscore the need to move beyond traditional architectures.

1.2 Analog In-Memory Computing for Efficient Neural Inference

The fundamental principle behind modern processors is the Von Neumann architecture, which consists of a Central Processing Unit (CPU)—divided into control unit and Arithmetic Logic Unit (ALU)—and a memory unit to store instructions and data. The memory is connected to the CPU through a bus that handles the transfer of data and instructions. This design introduces what is known as the *von Neumann bottleneck*, a throughput limitation caused by the shared communication path between the CPU and the memory. The continuous exchange of large amounts of data results in an intrinsic speed limit, since the processor does not work at its full capacity (the data transfer rate is lower than the actual rate at which the processor can work). Moreover, the physical separation between the processor and the memory leads to a significant amount of energy spent due to frequent data movement, which is relevant with respect to the energy needed for the computation.

The introduction of cache memory and the development of Near-Memory Computing (NMC) helped improve performance by reducing distance and, consequently, data transfer time. Nevertheless, the physical separation between the memory and the computation unit remains a limiting factor in the achievement of a better efficiency [9].

This is even more concerning when considering Machine Learning (ML) implementation, a key concept in the adoption of AI [2], making a hardware implementation of Deep Neural Network (DNN) a tough challenge.

An alternative approach draws inspiration from the human brain, where information is stored and processed within the same physical space, comprising neurons (computing elements) and synapses (memory elements). In a similar way, In-Memory Computing (IMC) proposes to perform some computational tasks directly where data are stored so that memory becomes now a computational memory unit: as a result, the computational efficiency is significantly improved [9]. The memory device can be based either on Dynamic Random access Memory (DRAM), Static Random Access Memory (SRAM) and Flash memory or on an

emerging class of memory devices known as Memristors.

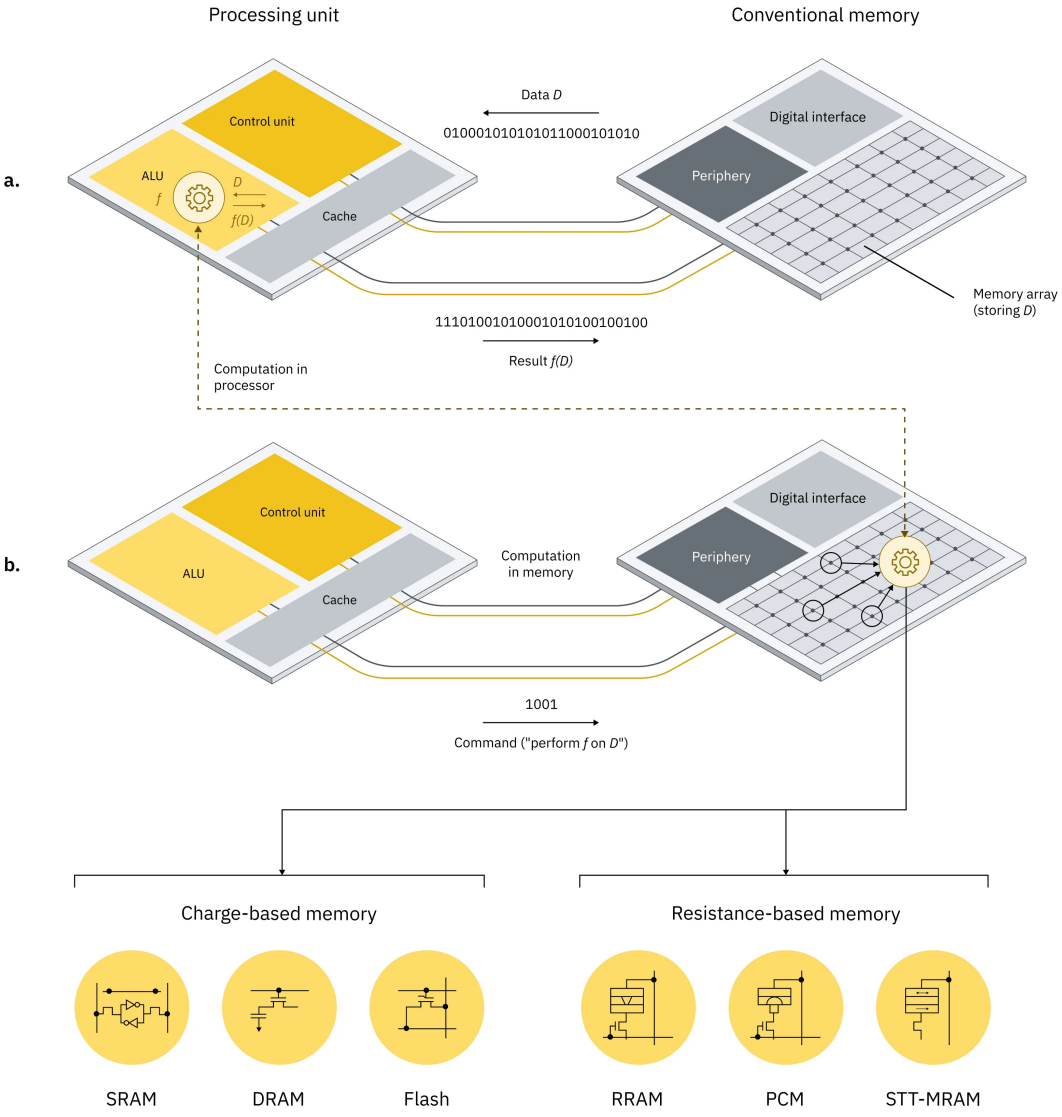


Figure 1.3: (a) Conventional computing system. (b) In memory computing system. [12]

The structure of the In-Memory compute module is very similar to the conventional one, with horizontal and vertical wires, respectively, Word Line (WL) and Bit Line (BL). However, some differences can be found in both the circuitry and the control logic [9]. For example, depending on the application, it may be necessary to activate multiple word lines in parallel or to accurately detect analog currents along the bit lines [9].

Even if there are different approaches to implement the memory device, some

drawbacks must be taken into account. In particular, SRAMs and DRAMs are volatile memories, meaning that an external power supply is needed to refresh stored data, while Flash Memories present high power consumption during writing and erasing operations. Therefore, memristor devices seem to be the most promising technology for implementing IMC. In particular, when analog Non-Volatile Memory (NVM) devices such as memristors are used to perform computation directly through physical laws, we refer to this approach as Analog In-Memory Computing (A-IMC) [3]. A-IMC typically relies on resistance-based NVMs integrated into crossbar memory arrays, with the goal of implementing the fundamental operation in neural networks: a hardware-accelerated, analog, and inherently approximate MVM [3]. In such architectures, inputs are applied as voltages along the rows (WLs), while the resulting output currents are collected along the columns (BLs), enabling parallel analog computation. Here, Ohm's and Kirchhoff's laws are exploited to perform MVMs directly, replacing the digital weight storage of conventional memories like DRAM or SRAM with analog conductance values [3].

As already described, in a simple fully connected neural network, a Multiply-Accumulate (MAC) operation is performed between an input vector of neuron excitations and a weight matrix to get a new vector at the output which represents the neuron excitations for the next layer. The same operation can be performed in memristor devices with electrical voltages and conductance values representing, respectively, neuronal activations and synaptic weights. MAC computation is conducted in a single step: the product $V \times G = I$ is calculated by encoding neuronal activations in voltage pulse amplitudes or lengths and, at the same time, the currents from every crosspoint are accumulated along the column. In this way, neural networks can be significantly accelerated with respect to digital systems, where the MAC is divided into two different sequential steps.

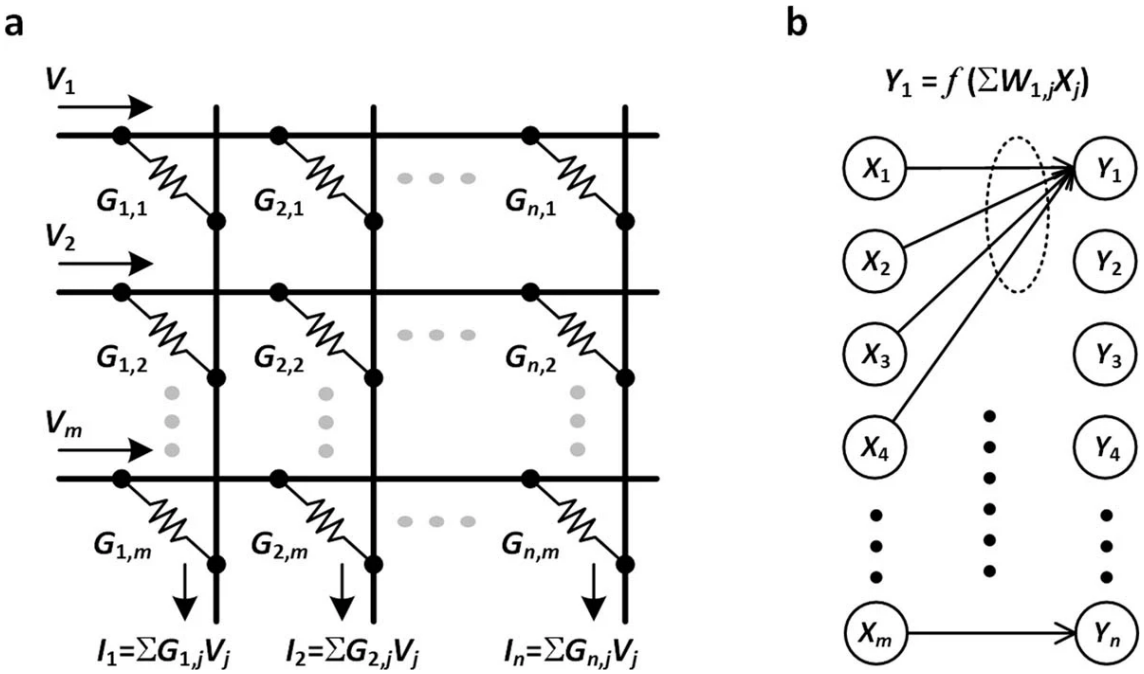


Figure 1.4: Fully connected Neural Network and Analog-based deep learning accelerator using a crossbar memory array. [15]

1.3 Memristive Devices for AIMC

In 1970, Leon O. Chua proved for the first time the existence of the memristor, as the fourth element of a basic two-terminal circuit. Indeed, considering the four fundamental circuit variables in the image 1.5, it is possible to derive the relationships between them, three of which were already defined and well known:

- The resistor, given by $dV = R\,dI$
- The capacitor, given by $dQ = C\,dV$
- The inductor, given by $d\varphi = L\,dI$

In [6], Chua theorized the memristor as the result of the only relationship that was still missing, that is the one between the flux and the charge. The name originates from the behavior of this element, similar to a nonlinear resistor with memory, hence "Memory-Resistor".[7]

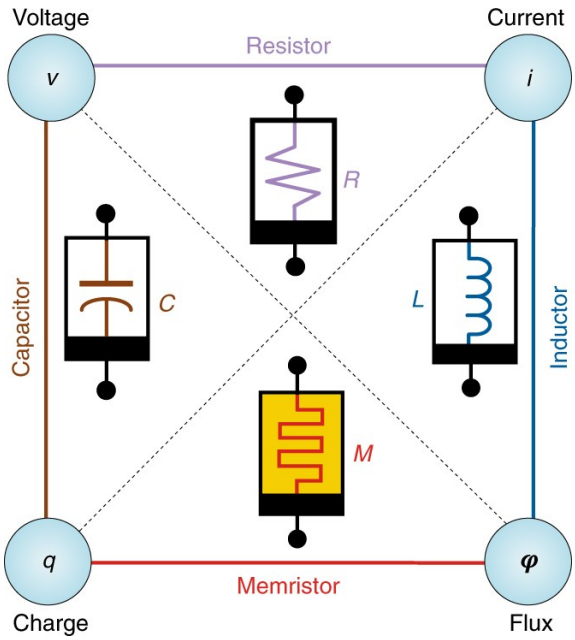


Figure 1.5: Resistor, Capacitor, Inductor, Memristor: the four fundamental two-terminal circuit elements. [7]

Memristors are characterized by pinched I-V hysteresis with respect to other passive circuit elements, which means that the current at a given voltage depends on the history of previous applied voltages.

The fundamental principle of these devices relies on the switching of two different resistance states: Low Resistance State (LRS) and High Resistance State (HRS). In particular, a transition from HRS to LRS corresponds to "*SET*" process and represents the storage of a digital "1". In contrast, with the opposite transition, from LRS to HRS, the memristor is "*RESET*", representing a digital "0".

The typical implementation consists of a 2D array known as CrossBar Array (CBA), where the memristor is integrated as a cross-point element. This simple two-terminal structure allows for extremely compact layouts and enables highly scalable integration into dense memory arrays. While traditional CMOS memories are based on three-terminal transistors and typically scale with an area of about $6F^2$, where F is the minimum feature size, memristors can be organized in CBAs with a theoretical scaling limit of $4F^2$ [14]. Furthermore, the flexibility in material selection and fabrication processes enables memristive devices to be integrated with existing CMOS technologies, especially in the Back-End-Of-Line (BEOL) stages [14].

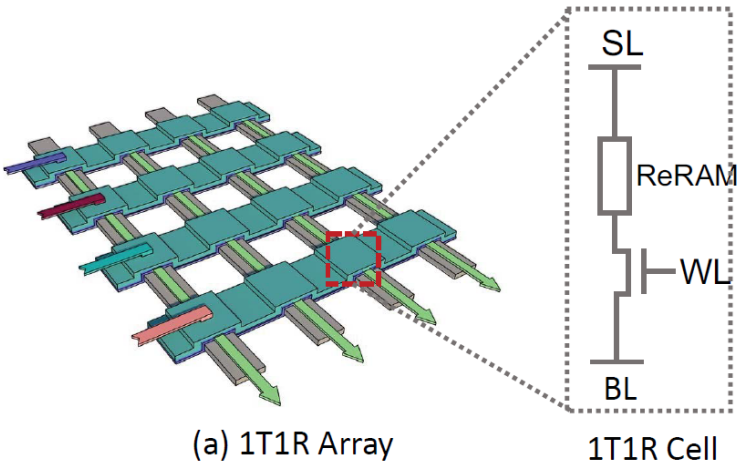


Figure 1.6: Physical implementation of the crossbar array. [28]

In ReRAM technology, each memory cell is realized with a 1T1R cell, a passive three-terminal device able to efficiently perform the MVM operation [2]. As shown in picture 1.6, the cell is composed of a memory element and a selector biased by a WL, which is used to open or close the circuit between the Source Line (SL) and the BL. With this implementation, the weight matrix is stored at the cross-points as the devices conductance, while the input vector is applied as analog voltage on the BLs, and the output current is accumulated on the BLs.
This structural simplicity and seamless CMOS integration are complemented by several functional advantages that further enhance the appeal of memristive technologies. In addition to their non-volatile nature and analog computing capabilities, memristors offer several key advantages that make them highly attractive for next-generation memory and computing systems. For instance, both read and write operations can be executed at significantly higher speeds compared to conventional memory technologies. This makes them suitable for applications that require low-latency access and high throughput.
While the term "memristor" is often used generically, various materials and physical mechanisms give rise to different classes of memristive devices, each with its own advantages, limitations, and operational modes. The most notable among them include Ferroelectric RAM (FeRAM), Phase-Change Memory (PCM), Resistive Random Access Memory (ReRAM), Magnetic RAM (MRAM) and are reported in Figure 1.7.

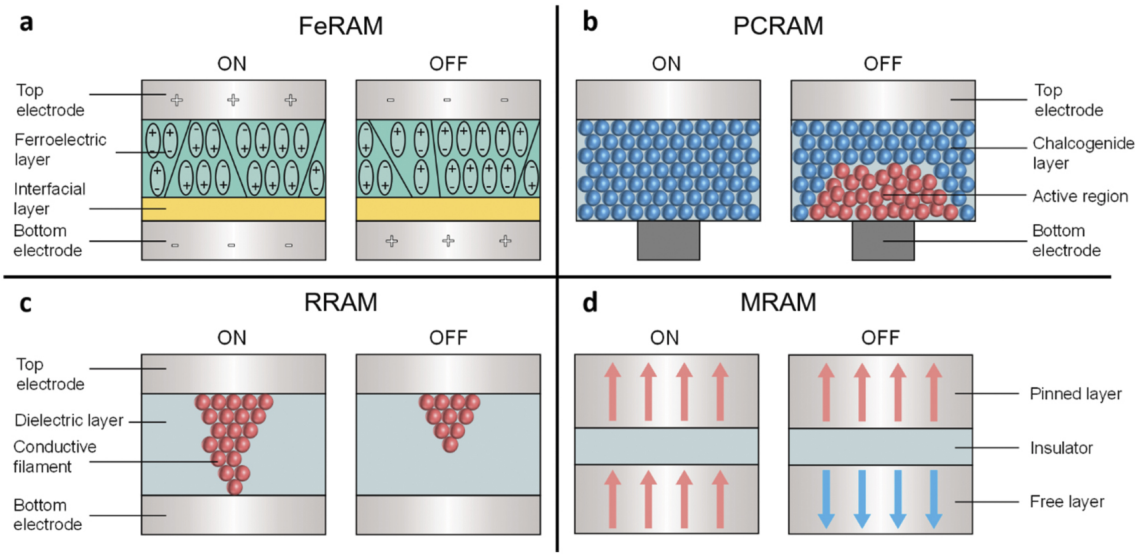


Figure 1.7: Overview of memristive technologies: FeRAM, PCM, ReRAM and MRAM. [4]

FeRAM The structure is based on a ferroelectric layer sandwiched between two metal electrodes and exploits the reversible polarization of ferroelectric materials to store data. When an external voltage is applied across the stack, the electric dipoles within the ferroelectric layer switch orientation, a process known as ferroelectric switching. This mechanism enables non-volatile data storage by encoding binary states through the direction of polarization. In some architectures, such as FeFETs, this polarization is also used to modulate the channel resistance, allowing for synaptic behavior and analog computing capabilities [4].

PCM Memristors based on phase-change concept store data by exploiting the resistance difference between two physical states of a phase-change material: a low-resistance crystalline phase and a high-resistance amorphous phase. The material is switched between these states using electrical current pulses, and the stored information is retrieved by measuring the device resistance [17]. Switching between the two phases is driven by Joule heating: to RESET the device (crystalline → amorphous), the material is heated above its melting point ($\approx 600C$) and then rapidly cooled, trapping it in the disordered, high-resistance state. The SET process (amorphous → crystalline) occurs by heating the material above the crystallization temperature ($\approx 400C$) and maintaining it for a sufficient time [25].

ReRAM These devices present a Metal-Insulator-Metal (MIM) structure, where a dielectric layer, often a metal oxide, is sandwiched between two metallic elec-

trodes. By applying voltage, the device undergoes a forming step that creates a conductive filament through the insulator, switching it into a LRS. The filament can be partially or fully disrupted (RESET) to return to a HRS. Depending on the switching mode, ReRAM can operate in bipolar (opposite polarities for SET and RESET) or unipolar (same polarity, different amplitude) modes.

MRAM In this case data are stored by manipulating the magnetization orientation of a ferromagnetic layer within a structure called a magnetic tunnel junction (MTJ). This junction consists of a thin insulating barrier placed between a fixed ferromagnetic layer (pinned layer) and a free ferromagnetic layer. The device shows low resistance when the magnetizations are parallel and high resistance when they are antiparallel.

1.4 ReRAM Operation and Noise Characteristics

Oxide-based ReRAM is a class of non-volatile memristive devices that modulate their resistance through voltage-induced redox reactions within a MIM structure [5]. The active layer is typically a metal oxide, most notably HfO_2, positioned between two metallic electrodes. This configuration allows the device to reversibly switch between a HRS and a LRS, enabling its use in both binary and analog memory applications.

ReRAM devices operate in two main switching modes:

- Unipolar switching, where both the SET (HRS → LRS) and RESET (LRS → HRS) transitions occur with voltage pulses of the same polarity but different magnitudes [26];
- Bipolar switching, where SET and RESET require opposite polarities, typically due to ionic drift or redox reactions localized at specific electrode interfaces [26].

To avoid permanent dielectric breakdown during the SET process in either switching mode, it is recommended to enforce a current compliance. This is typically achieved using a semiconductor parameter analyzer, or more practically, by integrating a selection device—such as a transistor or diode—or a series resistor within the memory cell [26].

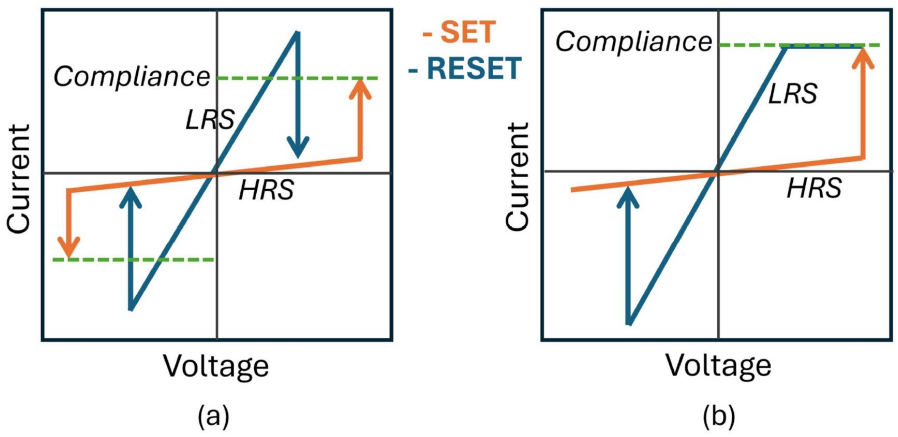


Figure 1.8: (a) Unipolar and (b) bipolar switching in ReRAM devices. [26]

Depending on the switching mechanism, ReRAM can be classified into filamentary and non-filamentary types. Filamentary devices rely on the formation and dissolution of conductive paths, often composed of oxygen vacancies, and usually require an initial forming step. Non-filamentary cells, instead, modulate the Schottky barrier at the electrode–oxide interface and generally operate without forming [5].

A common ReRAM configuration is the monolayer cell, typically based on a single HfO_x dielectric layer (Figure 1.9), where a series transistor is added for current compliance in a 1T1R configuration. However, this structure often suffers from asymmetry between SET and RESET operations, which limits its effectiveness in analog computing applications. To address this, a bilayer approach can be adopted, where two oxide materials are stacked between the electrodes (Figure 1.9). This architecture improves the symmetry of the switching characteristics, which is a crucial requirement for reliable and accurate weight updates during neural network training [27].

One strategy involves the introduction of a Thermal Enhanced Layer (TEL) on top of the primary oxide. The TEL is designed to have low thermal conductivity, which increases the local temperature during switching events [5]. This thermal confinement favors the formation of multiple, weaker conductive filaments, allowing for smoother and more gradual resistance modulation, an essential property for analog ReRAM behavior [22].

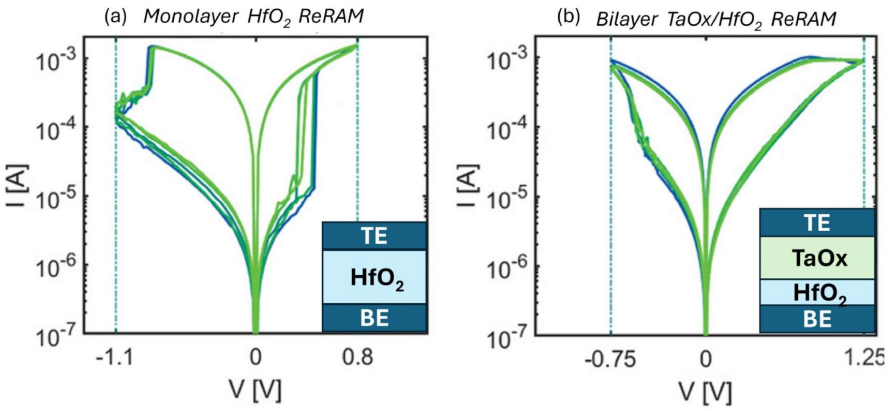


Figure 1.9: I-V sweep comparison of (a) monolayer HfOx ReRAM and (b) bilayer TEL/HfOx ReRAM. [22]

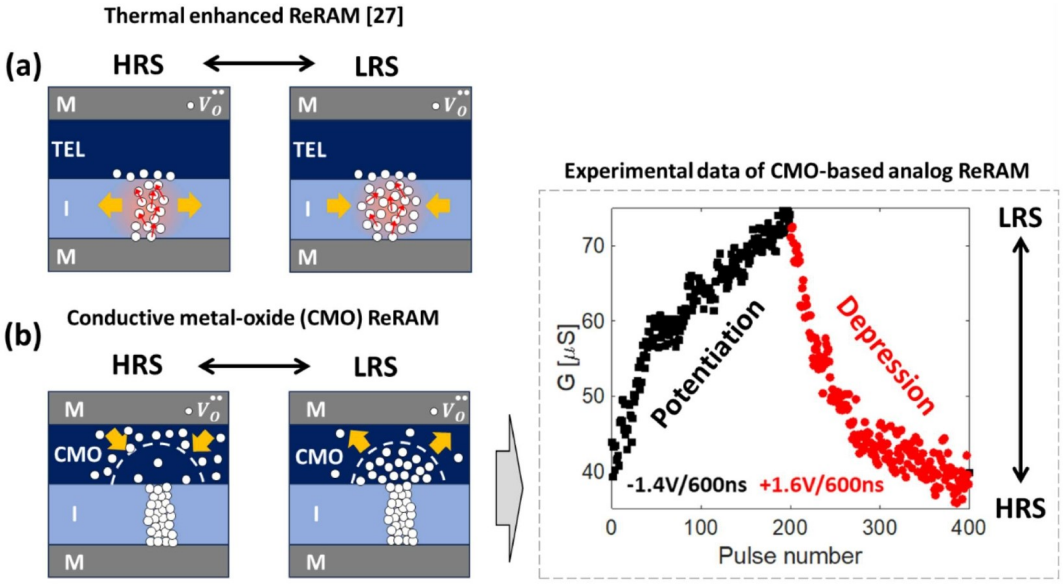


Figure 1.10: (a) Switching mechanism in TEL-based ReRAM, where a thermal barrier enhances filament broadening; (b) CMO-ReRAM structure with confined thermal/electric field. Right: corresponding analog conductance modulation under potentiation and depression pulses. [5]

Another effective approach is the integration of a Conductive Metal Oxide (CMO) layer into the MIM stack. Thanks to its higher electrical conductivity, the CMO helps confine the electric field and shifts most of the voltage drop onto the primary switching layer (e.g., HfO_x), thereby reducing the forming voltage. Furthermore, the CMO limits the self-accelerating effect often observed in the SET process, which in turn enhances device stability, endurance, and reliability [22]. These bilayer ReRAM stacks, compatible with Back-End-of-Line (BEOL) integra-

tion, offer nonvolatile analog switching, low power operation, and high scalability, making them strong candidates for in-memory computing applications.

This work focuses on CMO/HfO_x ReRAM devices, which have shown improved performance in terms of analog switching thanks to the optimized thermal and electrical properties of the CMO layer.

Despite the advances in ReRAM technology, various non-idealities at the device level still significantly impact the performance of IMC systems. These include device-to-device and cycle-to-cycle variations, Random Telegraph Noise (RTN), drift, and hard defects like stuck-open or stuck-short cells. Such imperfections can degrade the inference accuracy of neural networks when compared to their ideal software counterparts using the same synaptic weights [13].

The resistive array must be programmed with targeted weights before the neural network chip is deployed to a user environment. Hence, the conductance variability and retention of memristor devices is a significant reliability concern for their use in real applications. Deviations from the target conductance of a cell result in errors in MAC and MVM operations. Furthermore, these errors accumulate as signals propagate through the neural network layers, eventually causing the chip to malfunction [5]. Figure 1.11 shows how various noise sources and device-level affect the conductance values over time, from initial weight mapping to inference.

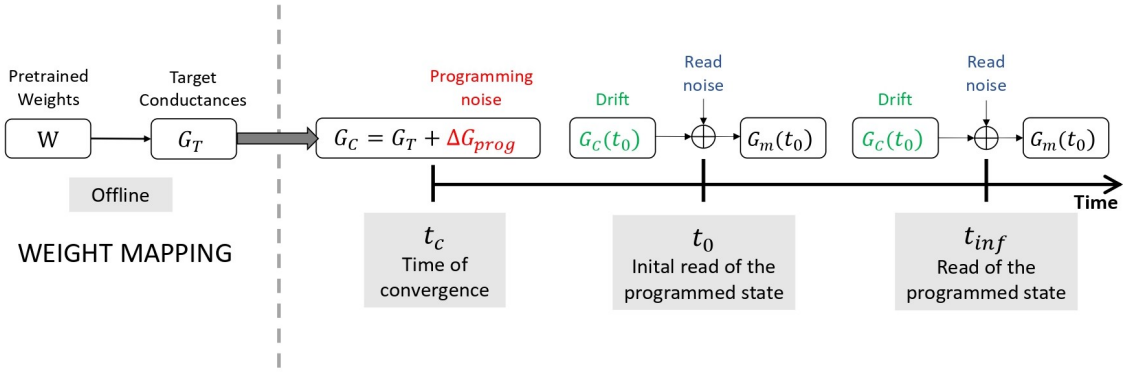


Figure 1.11: Temporal noise contributions affecting conductance levels in CMO-ReRAM based inference systems.

In memristor-based A-IMC inference accelerators, pre-trained and normalized neural network weights are first mapped into target conductance values G_T, a process referred to as weight mapping, typically performed offline. These target

conductances are then written into the hardware through an iterative procedure known as weight transfer, which continues until the programmed conductance converges to the target value within a predefined acceptance range [8]. Due to the analog and stochastic nature of memristive devices, this process introduces a programming error, resulting in a deviation from the target.

At the convergence time t_C, the device is assumed to have reached the target conductance. However, minutes after programming, memristive devices often exhibit conductance relaxation, a phenomenon in which the programmed state gradually shifts over time even in the absence of external stimuli. This drift in conductance, typically caused by the redistribution of ions or defects within the material, can lead to deviations from the intended weight values and thus degrade inference accuracy in deep neural networks [5]. Moreover, when the conductance is read, additional read noise is superimposed. Readout noise, being intrinsic to the device material, cannot be fully corrected by external circuits. This instability affects the reliability of write-verify methods, making it harder to accurately program and control conductance values [5].

Over time cumulative effects of drift and repeated read noise can further perturb the conductance level, leading to an effective synaptic weight that may differ significantly from the target. These non-idealities contribute to degradation in inference accuracy when compared to digital or software-based implementations.

In addition to device non-idealities, array parasitics also represent a significant challenge for IMC circuits. One of the major sources of circuit non-ideality is the parasitic wire resistance present along the rows and columns of the memory array. This resistance causes a current-resistance (IR) drop, which leads to voltage variations across the array during read and write operations. Such IR drops degrade the accuracy of analog computations by distorting the effective input voltages and output currents, ultimately impacting the inference precision of the neural network [13]. The severity of these parasitic effects typically increases with the array size and the complexity of the network, making it critical to carefully consider layout design, materials, and circuit-level compensation techniques to mitigate their impact.

In this thesis, the focus is placed on the realistic simulation of neural network in-

ference on CMO-ReRAM crossbar arrays, explicitly incorporating the effects of intrinsic device non-idealities and array-level parasitic phenomena. The first step was to develop and validate a statistical model for read noise based on experimental characterization. This read noise model was then integrated with existing models for programming noise and conductance relaxation, enabling hardware-aware simulations at crossbar array level that closely reflect the actual device behavior over time. Subsequently, the framework was extended to simulate neural network inference on three architectures — MLP, LeNet-5, and ResNet-32 — evaluating the impact of conductance relaxation and noise on inference accuracy over time.

2 Methods

This chapter aims to discuss the methods used in this thesis that support the analysis on the behavior of memristive devices for neural network inference.
All the simulations are performed by means of an open source Python toolkit developed by IBM and named IBM Analog Hardware Acceleration Kit (AIHWKIT) [21] to investigate and exploit the capabilities of Non-Volatile Memory (NVM) devices in the context of artificial intelligence. This simulator includes a statistical model for reproducing the behavior of ReRAM array during inference, taking into account the hardware characteristics, such as noise processes, analog-to-digital conversion resolutions, and other non-idealities. The model is calibrated using measurements from analog filamentary CMO-ReRAM arrays fabricated at IBM [8] [19] and it can be used only for inference tests, assuming that the weights have been pre-trained using dedicated software training algorithms.
In the following description, the noise sources included in the model are introduced. The publicly available noise model accounts exclusively for the effects of programming error and conductance relaxation. Therefore, based on this simulation environment, the statistical characterization from [19] for the read noise was considered in this work for further implementation. This addition completes the noise model landscape, enabling fully hardware-aware neural network simulations in the realm of AI inference. Building on this comprehensive framework, inference tests are conducted both at the crossbar array level and using complete neural network architectures.

2.1 Noise model

As already mentioned in the previous chapter, inference is directly affected by conductance stability. In the CMO-ReRAM array, three different sources of noise are considered and integrated during inference as illustrated in Figure 1.11. In this section, programming noise and conductance relaxation are described, whose models are already available from the analysis by Falcone et al. [8] and were orig-

inally characterized within a conductance range spanning from 10 to 90 μS.
Then, the methodology used to extract the read noise model is detailed. This model, derived from empirical measurements reported in [19], covers a conductance range approximately between 1 μS and 40 μS, and represents the missing component required to enable truly hardware-aware neural network simulations. The final model will be presented and discussed in the next chapter.

2.1.1 Programming error

The programming of ReRAM cells is typically carried out through a program-and-verify scheme, where the device is iteratively programmed and read until the resulting conductance falls within a predefined acceptance range around the target value. Multiple fluctuations in conductance can occur during the program-and-verify cycles before the target range is eventually reached [8]. The number of iterations required to reach the acceptance range is influenced by the inherent noise of the device as well as the width of the tolerance window set for programming accuracy. However, despite this iterative approach, the final programmed value does not exactly match the target conductance, resulting in a residual programming error. This error can lead to inaccuracies in MAC and MVM operations and is described as a Gaussian distribution with the standard deviation commonly referred to as *programming noise*.
Programming noise can be reduced, improving the programming accuracy, by selecting a narrower acceptance range. A fundamental trade-off has to be considered in this case since, with the reduction of the acceptance range, the number of iterations required for the convergence increases. In [8], Falcone et al. evaluated the trade-off taking into account two representative acceptance ranges, 0.2% and 2%. The standard deviation achieved is, respectively, less than 0.1μS and 1μS and depends linearly on the target conductance. The process of weight transfer is, therefore, almost ideal in analog ReRAM devices [8] for the selected conductance range.

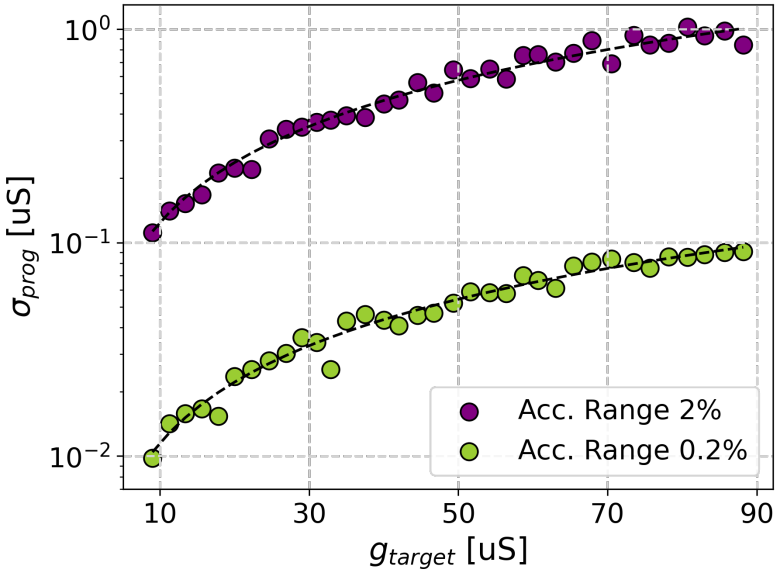


Figure 2.1: Experimental programming noise as a function of G_{target} for the two representative acceptance ranges. [8]

2.1.2 Conductance relaxation

A critical issue, instead, is represented by the *conductance relaxation* over time. The main cause behind this phenomenon is the Brownian motion of defects in the resistive switching layer [8]. An abrupt change in the conductance value is observed almost instantaneously, within a few seconds after programming, followed by a slower evolution over time.

The conductance level including the effect of the drift is described again as a Gaussian distribution with mean and standard deviation, respectively, decreasing and increasing with the logarithm of the time. Notably, its behavior is, instead, independent of the target conductance.

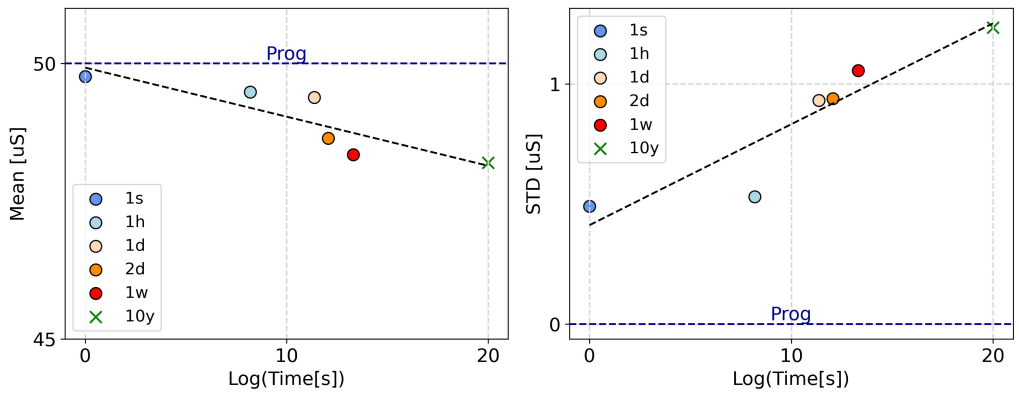


Figure 2.2: Time dependency of mean and standard deviation. [8]

Due to this phenomenon, the conductance values are expected to gradually decrease over time, and the experimentally observed temporal evolution is described as follows [8]:

$$\mu_{drift} = g_{prog} - 0.089 \cdot log(t)\ [\mu S] \tag{2.1}$$

2.1.3 Read noise

Finally, a full picture of ReRAM noise model can be obtained taking into account the *read noise*. This phenomenon arises from conductance fluctuations that cannot be avoided as caused by inherent material properties [5]. In particular, the conductance jumps repeatedly between two (or more) discrete levels and this behavior is known as Random Telegraph Noise (RTN). Besides a single-defect RTN, a general Flicker noise background can be observed to explain the read noise, which is attributed to a large ensemble of RTN signals [5].
An analytical expression for the read noise in ReRAM can be extracted from the analysis conducted by Lombardo et al. [19]. Although the same physical devices were used for all characterizations, the conductance range considered for this specific model $(1-40\mu S)$ is narrower than that employed for programming noise and conductance relaxation studies $(10-90\ \mu S)$.
Figures 2.3 and 2.4 provided in that study indicate a dependency of the read noise on both time and conductance level.

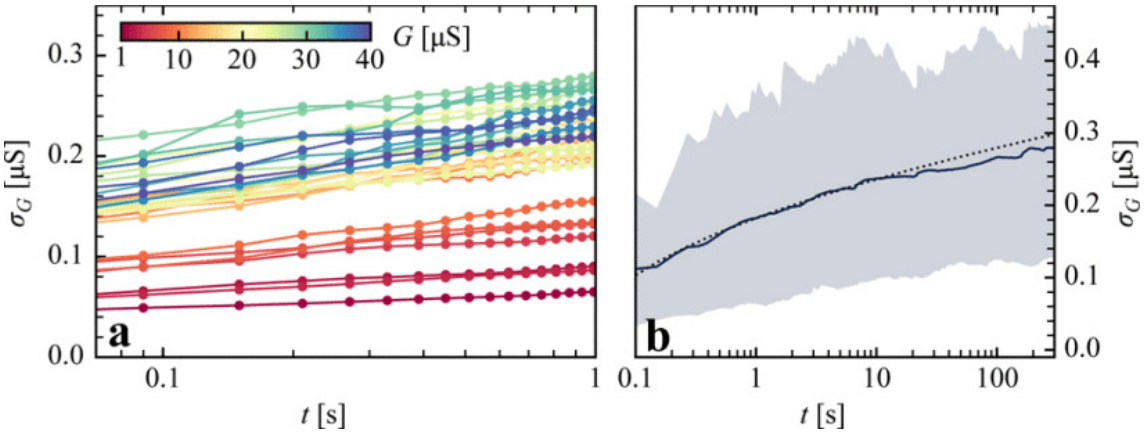


Figure 2.3: Read noise time evolution. [19]

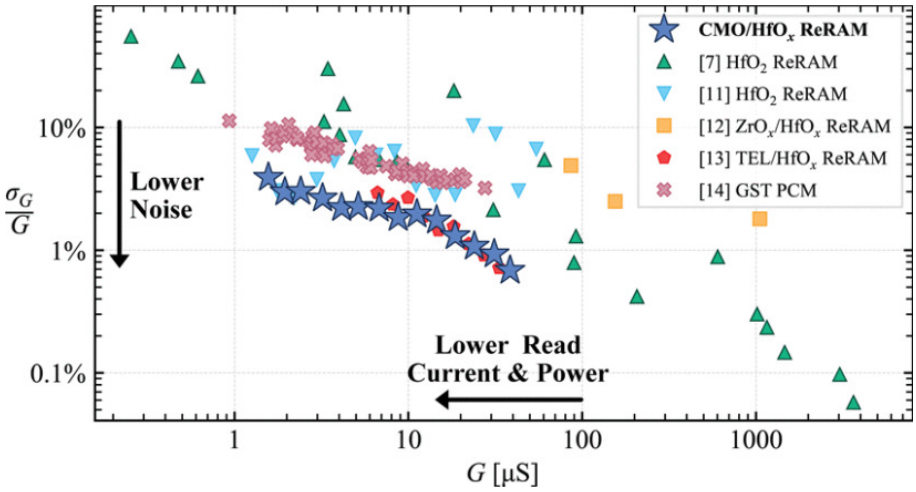


Figure 2.4: Average normalized conductance read standard deviation, compared with other devices found in literature. [19]

As already stated in [19], the standard deviation exhibits a logarithmic trend with respect to time. The latter indicates, in particular, the time elapsed between the programming and the MVM computation, and from now on it will be referred to as *inference time*. Moreover, the read noise also depends on the duration of the read pulse, which, in the cited characterization, is equal to $1MHz$.
The results in 2.4 show the average normalized standard deviation with respect to the conductance level compared to other devices found in the literature. From this analysis, the data of the CMO-HfO$_x$ ReRAM have been extracted to derive a model for the standard deviation of the read noise. To obtain an analytical expression that best describes the observed behavior, the extracted data points were fitted using the *curve_fit* function from the *scipy.optimize* package in Python. This function performs non-linear least squares minimization to fit a user-defined model to experimental data. In this case, a logarithmic function is chosen and the fitting algorithm estimates the optimal parameter K that minimizes the squared difference between the predicted and observed values. The quality of the fit was evaluated by computing the Root Mean Squared Error (RMSE) between the experimental data and the model output. This metric provides a direct measure of the average deviation between the observed and predicted values and is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.2}$$

where y_i are the observed values and $\hat{y}_i$ are the predicted values from the fitted model. The final fitted model is shown in the next chapter.

2.2 Crossbar array level simulations

All the described noise sources are considered when simulating the behavior of CMO-ReRAM devices, in order to assess the performance and understand the limits of this technology. In particular, these device-level effects are incorporated into a larger system by modeling them as the individual crosspoint elements in a crossbar array in the AIHWKIT. This allows the investigation of how such non-idealities influence the overall MVM operation at array level, bridging the gap between physical device behavior and system-level performance.
Simulations are, thus, conducted on a single-crossbar array configuration using the AIHWKIT toolkit mentioned before [21]. This allows to isolate and understand the impact of both device-level non-idealities (e.g., conductance relaxation) and array-level effects (e.g., IR drop) on MVM accuracy.
The digital weights follow a normal distribution $\mathcal{N}(0, 1)$, with zero mean and unit standard deviation, and are then normalized into a range $[-1, 1]$, with -1 corresponding to the HRS, i.e. g_{min}, and $+1$ to the LRS, i.e. g_{max}. This assumption is well established in the literature on neural network initialization. Each weight is, then, programmed into a single device within the range $(9, 88.19)$ μS.
The simulations are conducted starting from an input vector with a uniform distribution in the range $[-1, 1]$. Moreover, two additional hardware characteristics have to be taken into account when considering a crossbar configuration: the effect of quantization and parasitic wire resistance. Since the aim is to isolate and analyze the effect of analog non-idealities on MVM, both the input DAC and output ADC are configured with a resolution of 32 bits: this high-precision setup allows to avoid the influence of quantization noise making it clearer to observe the effects of other non-idealities. Among these, parasitic wire resistance represents a major source of non-ideal behavior at array level and its impact is one of the objects of this thesis.
When a current flows through the resistive lines of the array, a portion of the volt-

age drops along the path due to line resistance. This results in a reduction of the effective voltage across the memory cells, potentially causing inaccuracies in the MVM. The magnitude of this parasitic wire resistance strongly depends on the underlying technology. In larger or older technology nodes (with feature sizes of the order of micrometers), the interconnect wires are typically longer and wider, resulting in higher resistance values. Conversely, in scaled technologies, smaller wires are used, which reduces parasitic resistance. For instance, in the IBM Hermes Project chip design [16], implemented using a 14 nm technology node, the wire resistance is reported to be around $0.15\,\Omega$. In this case, the devices from Lombardo et al. [19], fabricated using a 130 nm technology node, exhibit a wire resistance of approximately $0.35\,\Omega$.

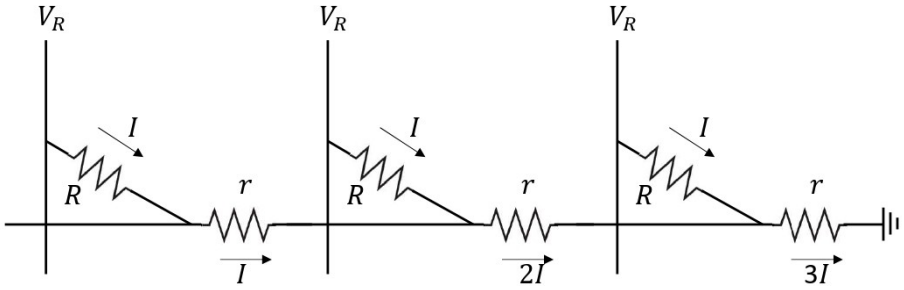


Figure 2.5: Parasitic voltage drop across a three-cell row with memory resistance R and wire resistance r.

Analytically, the effect of the IR drop on the input voltage increases with the number of input rows of the array. Assuming the same current for each device, the overall voltage due to the parasitic resistance across the wire is

$$rI + 2rI + 3rI + \cdots + NrI = rI \cdot (1 + 2 + 3 + \cdots + N) = rI \cdot \frac{N(N+1)}{2} \quad (2.3)$$

where N corresponds to the number of bit lines [13]. For larger sizes, the expression can be approximated by $rI \cdot \frac{N^2}{2}$. Therefore, a critical drawback is that, as the size of the crossbar array increases, the resistance of the interconnects grows, amplifying the effect of the IR drop.

An important parameter that determines the severity of this phenomenon is the product $G_{max} \cdot R_{wire}$. As a matter of fact, a higher G_{max} implies a lower minimum resistance of the devices, meaning that, when a voltage V is applied across

the array, a larger current flows through the interconnects. This, in turn, leads to a higher voltage drop across the parasitic resistance of the wires and electrodes. Therefore, the product $G_{max} \cdot R_{wire}$ quantifies how much the array conductance amplifies the effect of wire resistance, directly influencing the IR drop. If this product is too large, the IR drop becomes significant and can substantially degrade the accuracy of analog MVM. Therefore, the current I in Equation 2.3 is strongly determined by the conductance range of the devices in the crossbar array.
CMO-ReRAM devices exhibit a maximum conductance $G_{max} = 90\mu S$, resulting in relatively high current levels. To provide a concrete example, in an array with 128 input lines, wire resistance $r = 0.35\ \Omega$, maximum device conductance $G_{\text{max}} = 90\ \mu S$, and read voltage $V_{\text{read}} = 0.2\ V$, the current through each device is:

$$I = V_{\text{read}} \cdot G_{\text{max}} = 0.2\ \text{V} \cdot 90\ \mu\text{S} = 18\ \mu\text{A}.$$

The voltage drop across the wire can be estimated as:

$$\Delta V = rI \cdot \frac{N^2}{2} = 0.35\ \Omega \cdot 18\ \mu\text{A} \cdot \frac{128^2}{2} \approx 51.4\ \text{mV},$$

which corresponds to approximately 25.7% of the applied read voltage. This example highlights how the IR drop can significantly affect the accuracy of analog MVM computations even in moderately sized arrays.
In the next chapter, the impact of this effect will be discussed, considering a parasitic resistance of $0.35\,\Omega$, as previously specified for the devices used in this study. To this aim, the performances of the device under discussion are evaluated in a time interval from $0\ s$ to $10\ years$ taking into account the MVM error based on the Euclidean distance between two sets of vectors:

$$\epsilon_{\text{MVM}} = \frac{\|y_i - \hat{y}_i\|_2}{\|y_i\|_2} \tag{2.4}$$

where y_i is the result of the analog MVM, while $\hat{y}_i$ is the real dot product $W \cdot x_i$. The analysis is conducted for the following crossbar sizes:

- 64x64
- 128x128

- 256x256
- 512x512.

Besides of the IR drop, the relaxation effect is studied at crossbar array level. As explained in the previous section, Analog ReRAM devices are characterized by an abrupt relaxation towards the HRS immediately after programming, showing a decelerating trend with respect to time. When performing in-memory Matrix-Vector Multiplications the mean shift (conductance decrease) can systematically reduce the output. In order to analyze this non-zero mean drift, the weight distribution is observed for a 64x64 crossbar array for two distinct time instants - immediately after programming and after 10 years - and under two scenarios: with the experimentally observed mean of the conductance relaxation according to Equation 2.1 [8] and with the mean set to 0. This assumption is justified by the fact that the mean of the conductance relaxation is independent of the conductance and can be effectively corrected at the programming step.
To complete the analysis, the MVM error is also computed according to Equation 2.4 varying both the array size and the value of the non-zero mean drift. The resulting heatmap helps identify the region of operating conditions, in terms of crossbar size and non-zero mean drift, that ensures a tolerable MVM error over a 10-year period.

2.3 Neural Network Implementation and Evaluation Strategy

Finally, the analysis is extended to conventional neural network models performing standard machine learning tasks, such as image classification, to assess how inference is affected when operating with analog ReRAM crossbar architectures. This step aims to understand how the accuracy of pre-trained networks degrades when realistic hardware effects—such as device noise, IR drop, and quantization from ADC/DAC interfaces—are introduced during inference. The transition from array-level analysis to full-network evaluation allows a more comprehensive understanding of how hardware imperfections influence practical inference tasks.

Image classification is selected as the reference task since it is one of the most established benchmarks in machine learning, with a wide range of neural network architectures extensively tested, both in digital and analog implementations. As such, it provides a solid ground for comparison and facilitates the evaluation of accuracy degradation due to hardware effects. In particular, the MNIST and CIFAR-10 datasets are used, representing standard classification tasks with increasing levels of complexity and commonly adopted for benchmarking in both software and hardware-aware studies.

- MNIST consists of seventy thousand 28x28 pixels grayscale images of handwritten digits. It is commonly used for simple image classification tasks and the maximum obtained accuracy in literature is 99.87%.

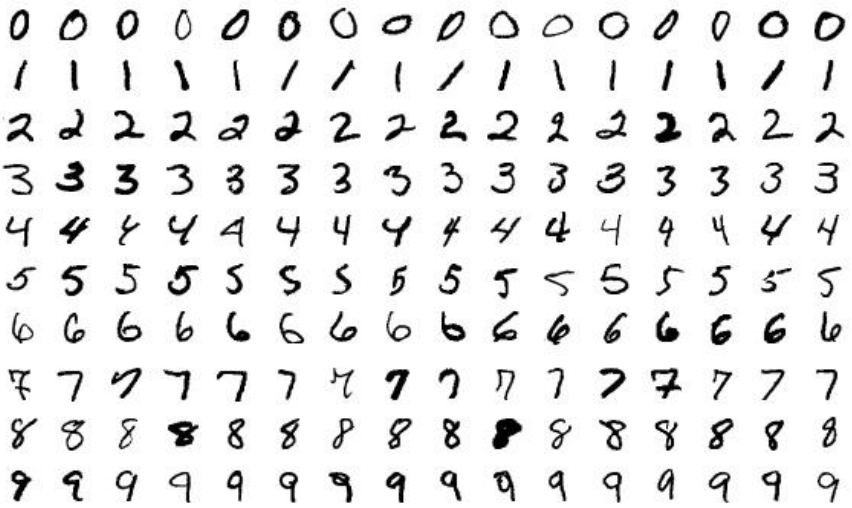


Figure 2.6: MNIST dataset.

- CIFAR-10 includes sixty thousand 32x32 pixels RGB color images. The images are divided into 10 different classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks), making the adoption of this dataset more challenging. In this case, the best accuracy achieved in prior studies is 99.5%.

Figure 2.7: CIFAR-10 dataset.

Three neural network models are selected for analysis based on their widespread adoption and well-established benchmarking results in the literature. The simplest model considered is the Multi-Layer Perceptron (MLP). In addition, Convolutional Neural Network (CNN) architectures such as LeNet-5 and ResNet are included, as these are commonly used for image classification tasks. By employing convolutional kernels, CNNs capture image features more efficiently than traditional fully connected networks, often resulting in improved classification accuracy. The inclusion of these architectures allows for a comprehensive evaluation across different model complexities.

Inference is performed on neural network models pre-trained using Stochastic Gradient Descent (SGD). MLP and LeNet-5 models are trained on the MNIST dataset with a batch size of 8 and a learning rate of 0.01. In contrast, the ResNet model is trained on the more complex CIFAR-10 dataset, using a batch size of 128 and a learning rate of 0.05. Simulations are conducted by adopting the CMO-ReRAM model included in the AIHWKIT, therefore considering the noise model mentioned at the beginning of this chapter, a wire resistance equal to $0.35\,\Omega$, $G_{max} = 88.10\,\mu S$, and input and output resolutions, respectively, equal to 6 and 8 bits. The accuracy in percentage is evaluated at five different time steps: immediately after programming, after $1\,s$, $1\,h$, $1\,day$ and $10\,years$. The evolution of the accuracy over time allows to observe the effect of conductance relaxation, and the same experiments are conducted setting the mean of the conductance relaxation equal to 0, in

order to make a comparison and understand the impact of this phenomenon on long-term inference performance.
Moreover, to account for variability, five independent inference runs are performed for each time step, and, at the end, the average accuracy is computed. The digital floating-point model is used as a reference to highlight the effects of deploying the network on analog hardware.
The following paragraphs focus on the structure of the mentioned models, while a clear explanation on how these architectures are translated into hardware will be detailed in the next chapter.

2.3.1 Multilayer Perceptron

The Multi-Layer Perceptron (MLP) is a fully connected feedforward neural network in which each neuron in a given layer is connected to every neuron in the subsequent one. Each connection is associated with a weight that is learned during the training process. The model was pre-trained on the MNIST dataset, and its architecture is summarized in Table 2.1.

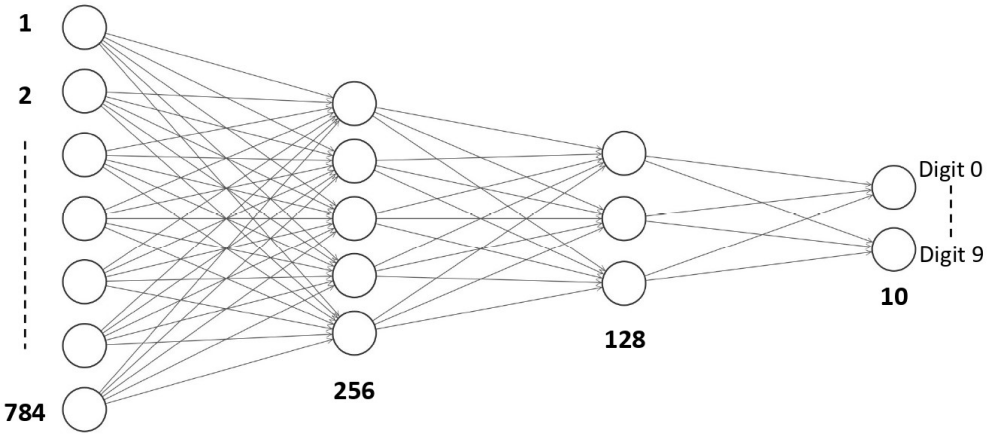


Figure 2.8: Visual representation of the MLP architecture.

	Layer	Output size	Activation
1	Fully connected	784x256	–
2	Fully connected	256x128	–
3	Fully connected	128x10	LogSoftMax

Table 2.1: Layer-by-layer architecture of the MLP model.

2.3.2 LeNet-5

LeNet-5 is one of the first convolutional neural networks for handwritten character recognition. Unlike fully connected neural networks, CNNs use a convolutional layer to extract some features from the input through a set of learnable filters. To be more precise, a small matrix, called a filter or kernel, is applied to the input data and slides along the input matrix performing element-wise multiplications and summing the results. This operation, known as convolution, generates at the output a feature map which shows where the detected pattern is. Often, each convolutional layer of the network has more than one filter, each of them learning a specific feature of the input image, and the feature maps represent the different types of feature extracted at various spatial locations. Two important concepts should be introduced when talking about a convolutional layer:

- Stride: it defines how many pixels the filter moves at each step. With a larger stride, the output dimensions are reduced, but in this case $s = 1$ is chosen to preserve more spatial information.

- Padding: it controls the output size by adding extra pixels (typically zeros) at the edges in order to preserve the dimension of the input image. If no padding is applied, the size of the feature map at the output is reduced with respect to the input.

At the end, the convolutional layer gives at the output a feature map with the following dimension:

$$\text{Output size} = \frac{\text{Input size} + 2 \times \text{Padding} - \text{Kernel size}}{\text{Stride}} + 1$$

A "$+1$" is added to take into account the initial position of the filter at the beginning of the input feature map: without this adjustment, only the filter steps would be counted, while the initial placement of the filter has to be considered as one valid step.

Usually, after the operation of convolution, a pooling layer is applied to the feature maps to reduce the dimensions of the data, though retaining the most relevant information. Two different kinds of pooling can be used, Max Pooling or Average

Pooling: the first one selects the maximum value within each pooling window, while the second one computes the average of the values in the window. In this case, Max Pooling is adopted as it preserves the most prominent information, making the network more robust to small translations.
Finally, non-linearity is introduced by applying an activation function such as ReLU or LogSoftMax.
In the final part of the architecture, fully connected layers are typically used to make a prediction, connecting all features together.
A complete implementation of LeNet-5 is detailed below.

Layer	**Output size**	**Activation**
Input image	28×28×1	–
Convolutional (5×5 kernel, 16 filters, stride 1, no padding)	24×24×16	–
Max pooling (2×2 window, stride 2)	12×12×16	ReLU
Convolutional (5×5 kernel, 32 filters, stride 1, no padding)	8×8×32	–
Max pooling (2×2 window, stride 2)	4×4×32	ReLU
Flatten (4x4x32 → 512)	512	–
Fully Connected (512 → 128)	128	ReLU
Fully Connected (128 → 10)	10	LogSoftMax

Table 2.2: Layer-by-layer LeNet-5 architecture.

In addition to overall accuracy evaluations, for the LeNet-5 network, a per-layer error analysis is also performed, quantifying the deviations introduced at each key stage of network inference. In particular, the error metric used for this study is the same adopted when working at array level, mentioned in the previous paragraph and summarized in Equation 2.4. Errors are collected for the two convolutional layers, the two fully connected layers, and the final log-softmax output layer. Intermediate operations, such as pooling and activation functions, are excluded from this analysis as they do not involve analog matrix-vector multiplications and are thus not significantly impacted by ReRAM-related non-idealities.

Also in this case, the simulations are performed for five different time instants (immediately after programming, after $1\,s$, $1\,h$, $1\,day$ and $10\,years$) and under two scenarios: with the experimentally observed coefficient used to evaluate the mean of the conductance relaxation (equal to -0.08 according to Equation 2.1) and with the mean set to 0. Finally, a third simulation is carried out, varying the value of the coefficient just mentioned in a range between $[0, -0.1]$, while considering the inference time equal to 10 years.

2.3.3 ResNet

The term ResNet stands for "Residual neural network" and indicates a deep learning neural network where each layer is designed to learn residual functions, that is, the difference between the input and the output, rather than the actual output. The main component of a Residual Network is the residual block, whose input is bypassed over the convolutional layer and added to the output of the residual block. Therefore, the output of the residual block $H(x)$ is the sum of the residual mapping learned by the network $F(x)$ and the input of the block itself.

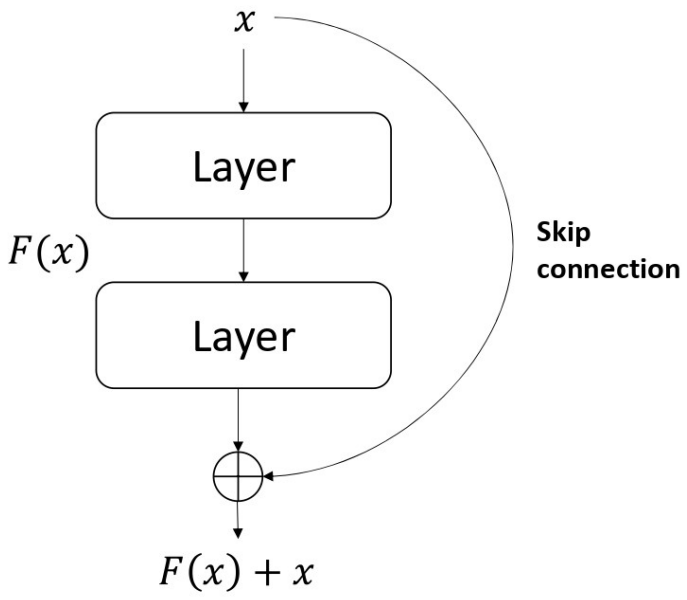


Figure 2.9: Residual Block.

The operation is performed by a shortcut (or skip) connection with identity mapping. This technique has great advantages during backpropagation. Indeed, a critical issue when training a neural network is represented by the vanishing

gradient problem: as the gradients are propagated backward through the layers of the network, they become progressively small, causing small weight updates and thus, preventing the layers closer to the input from learning. The problem is even more serious when dealing with neural networks consisting of many layers. By using a residual network, since the gradient can propagate directly through the shortcut connection, non-linear intermediate layers are bypassed and early layers receive stronger gradient signals, enabling more effective learning.

A detailed description of the model for ResNet-32 used in this work is presented in table 2.3.

The input data is a 3-channel 32x32 image from the CIFAR10 dataset, which is processed in the standard ResNet network composed of an initial convolution layer followed by three stages of residual blocks. Each stage uses five basic blocks, with output channel sizes, respectively, of 16, 32 and 64; thus, the network uses a total of 15 residual blocks. The residual blocks in each stage consist of two convolutional layers with kernel size 3x3 and a shortcut connection that performs identity mapping or downsampling when the input and output dimensions do not match. At the end, a global average pooling layer reduces the dimensions before passing the output to a fully connected linear layer to perform the final classification.

Layer	**Output Size**	**Details**
Input	$3 \times 32 \times 32$	RGB image
Conv1	$16 \times 32 \times 32$	3×3 conv, 16 filters, stride 1, padding 1
Layer1 (5 blocks)	$16 \times 32 \times 32$	5 × BasicBlock (identity shortcut)
Layer2 (5 blocks)	$32 \times 16 \times 16$	1st block: stride 2 (downsampling) then 4 × BasicBlock
Layer3 (5 blocks)	$64 \times 8 \times 8$	1st block: stride 2 (downsampling) then 4 × BasicBlock
AvgPool	$64 \times 1 \times 1$	Global average pooling
Flatten	64	Flatten the $1x1$ feature map
Fully Connected	10	Dense layer for classification

Table 2.3: Layer-wise structure of the ResNet-32 architecture.

2.3.4 Integrating Hardware-Aware Training

Although the mapping of neural network weights into conductance levels seems straightforward, several sources of non-idealities have to be considered. Indeed, the MVM performed by the crossbar array can be written as follows:

$$\mathbf{y} = \text{ADC}[W_r \cdot \text{DAC}(\mathbf{x}) + \text{noise}]. \quad [11]$$

The input vector $\mathbf{x}$ is first converted into analog by a DAC and then multiplied by the weight matrix W_r, which represents the conductance values stored in the crossbar array. This multiplication is affected by analog noise due to hardware defects and converted again to digital by an ADC.

In addition to the noise introduced by the analog computation and the one related to the DAC and ADC, some deviations of the conductance values from the original weights of the network should be taken into account. All these hardware non-idealities result in reduced network accuracy, making the analog approach impractical, in particular for large scale networks.

One effective strategy to mitigate this problem is represented by Hardware-Aware (HWA) Training: the idea is to train the network on a digital accelerator in a way that is aware of the hardware characteristics. Thus, some device non-idealities, such as programming noise and IR drop, are introduced in the forward pass so that the neural network itself learns how to be resilient to them, while learning meaningful features for accurate classification. However, the model adapts to instantaneous non-idealities but long-term effects like conductance drift are not explicitly modeled. Additional white Gaussian noise is typically injected to emulate device variability. This makes the model more resilient to stochastic deviations, regardless of their origin. However, while this approach partially captures the standard deviation associated with conductance relaxation, it does not account for systematic changes caused by the non-zero mean drift, that can have a significat impact in long-term inference accuracy.

The next chapter of this work presents simulations of the previously discussed neural networks, trained with the HWA Training methodology and their inference accuracy is, then, compared to that of analog models obtained from standard

floating-point training, without incorporating hardware-aware optimization. The simulations are carried out considering all the specifications already described in this chapter.

3 Results

In this chapter, the results obtained from the analysis and simulations conducted throughout this work are presented and discussed.
The first section aims to describe the model for read noise derived from electrical characterization, as introduced in 2.1. CMO-ReRAM devices are then simulated, starting at the array level and finally within neural network architectures, to assess how device-level imperfections and additional crossbar characteristics affect the inference accuracy. In particular, crossbar array level simulations are carried out to investigate limitations such as IR-drop and conductance relaxation, providing insights into trade-offs between array size and device variability. Subsequently, the performance of different neural network models detailed in 2.3 — MLP, LeNet-5, and ResNet-32 — are evaluated through hardware-aware simulations to investigate the long-term behavior of ReRAM crossbar arrays in large-scale neural networks and to assess the improvements introduced by hardware-aware training.
All the analysis are conducted following the procedures outlined in the previous chapter.

3.1 Read Noise Modeling and Validation

In order to have a realistic model that is able to reproduce the behavior of ReRAM devices considering hardware non-idealities, three different noise sources should be modeled and taken into account. Two of them, namely programming noise and conductance relaxation, are already provided by Falcone et al. in [8], and have been detailed in Section 2.1.
In the same context, the third noise source —read noise — was introduced based on the electrical characterization provided in [19]. In this study a statistical model for the read noise is derived, starting from those experimental results.
As already presented in [19], the standard deviation of the read noise exhibits a logarithmic trend with respect to the inference time, which is shown in Figure 2.3. Figure 2.4, instead, presents the experimental relationship between the standard

deviation, normalized to the corresponding conductance value and expressed in percentage, and the conductance level, up to approximately $37\ \mu S$. It is possible to notice that as the value of the conductance increases, the normalized standard deviation decreases between 2% and 0.2%. Therefore, an initial observation, due to the logarithmic scale, is that σ_G exhibits an approximately constant behavior at higher conductance levels. The dependency was extracted by fitting the measured data, which captured a logarithmic trend in the range of interest $(8 \sim 40)\ \mu S$, as illustrated in Figure 3.1.

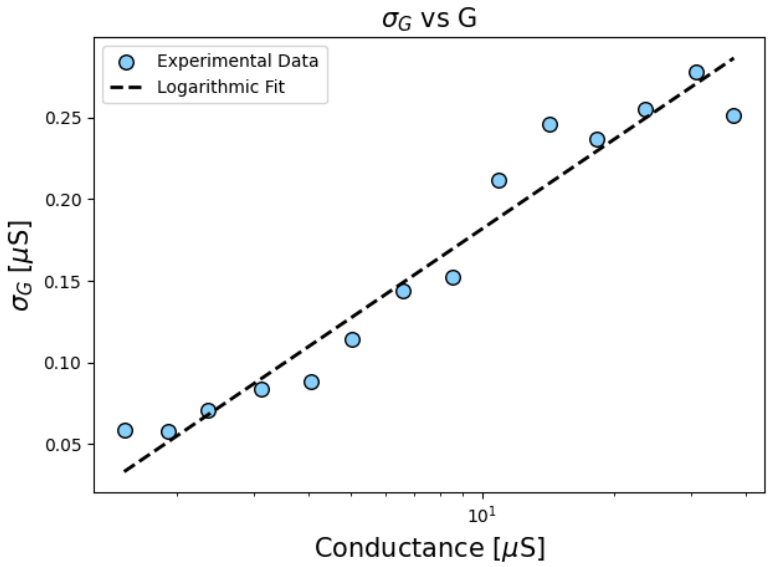


Figure 3.1: Fitting of the read noise standard deviation as a function of the conductance level, based on experimental measurements from Lombardo et al.

Based on this analysis, the read noise was modeled as a zero-mean Gaussian distribution with standard deviation

$$\sigma_G(G_{\text{drift}}, t_{\text{inf}}) = [K \cdot \log_{10}(G_{\text{drift}})] \cdot \sqrt{\log(f_{\text{max}} \cdot t_{\text{inf}})}$$

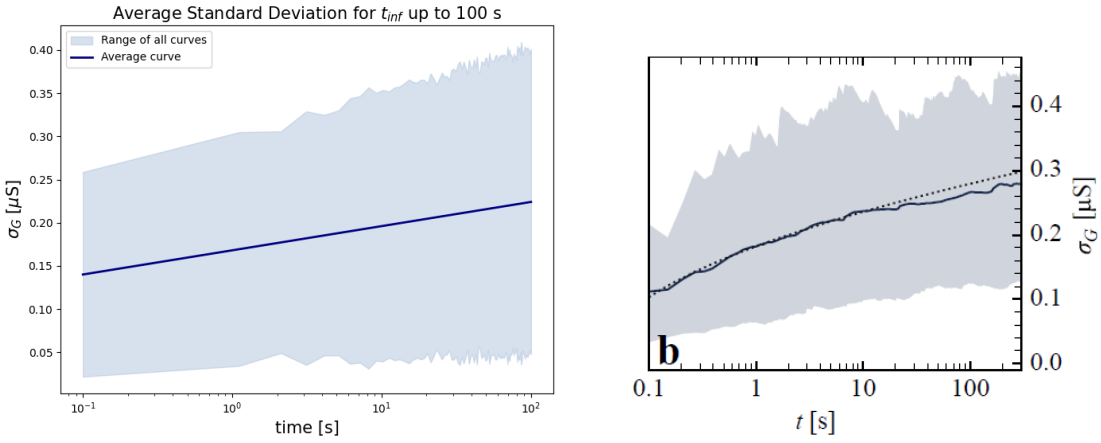
In the equation:

- K is the coefficient obtained from the curve fitting procedure and is equal to 0.0277. This parameter minimizes the fitting error, resulting in a RMSE of 0.0187.
- G_{drift} represents the conductance value at the time of measurement, i.e. the drifted conductance.

- $f_{max} = \frac{1}{2 \cdot t_{tread}}$ indicates the dependency on the read pulse and, in this case, is equal to $1\ MHz$.

- t_{inf} is the inference time, that is the time elapsed between the programming of the device and the MVM calculation.

To reproduce the evolution with respect to time and compare this model with the experimental data, σ_G was evaluated for t_{inf} up to $100\ s$ for the same conductance range between $8\ \mu S$ and $40\ \mu S$ and the curve of the average standard deviation is represented in Figure 3.2a. The outcome demonstrates a strong correlation with the experimental observations, supporting the validity of the proposed model.



(a) (b)

Figure 3.2: Time evolution of average read noise from: (a) statistical model; (b) experimental data [19].

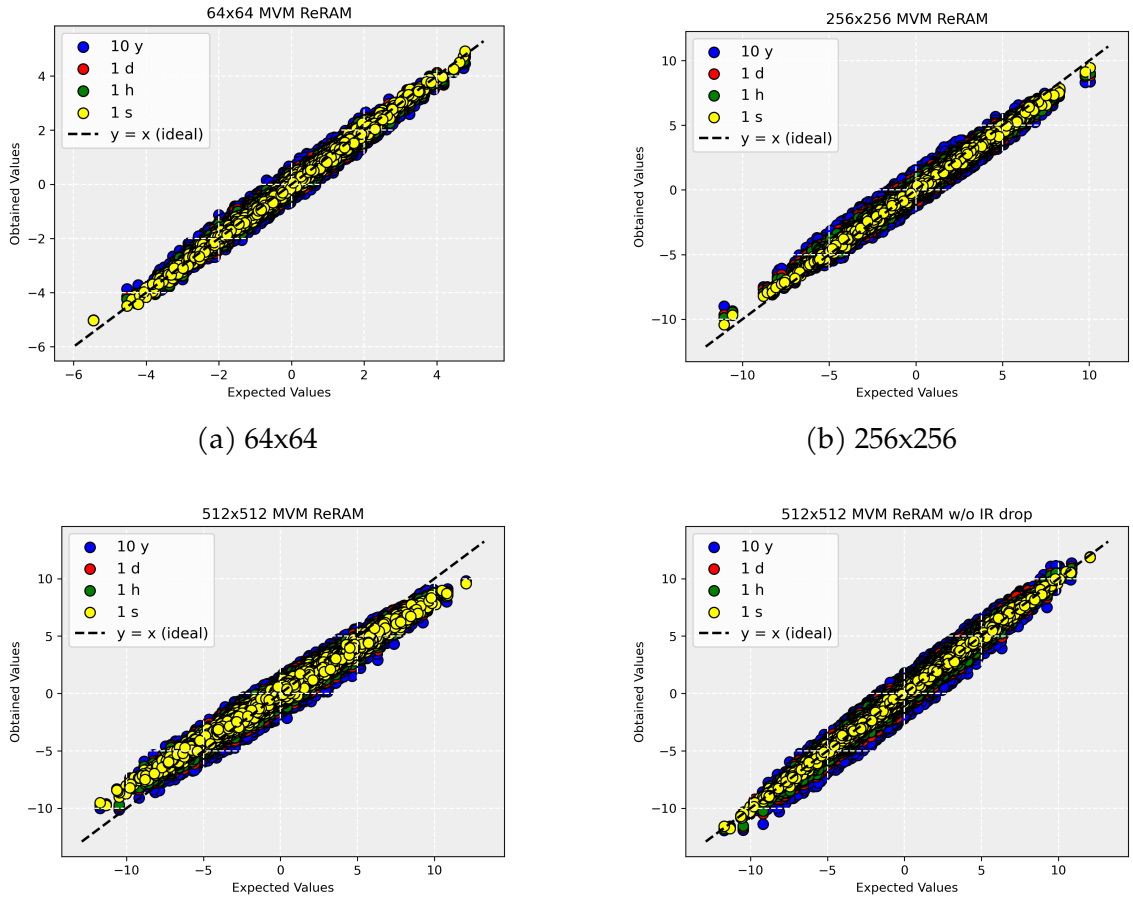
3.2 Impact of Non-Idealities at Array Level

Once obtained the expression for read noise, the complete noise model for CMO-ReRAM is used to simulate the behavior of this technology at the crossbar-array level. As described in section 2.2, this analysis aims to evaluate how different types of non-idealities affect MVM operations. To this end, two main categories are considered:

- Device-level non-idealities, such as conductance relaxation, which depend on the physical behavior of individual memory cells and typically evolve over time.

- Crossbar-level non-idealities, such as IR drop, which arise from the resistive nature of interconnects and increase with the size of the array.

First of all, simulations are performed varying the array size in order to assess the limitations imposed by IR drop. In particular, the obtained and expected outputs are compared for different dimensions of the crossbar array, respectively 64x64, 256x256 and 512x512.



(a) 64x64

(b) 256x256

(c) 512x512

(d) 512x512 with IR drop disabled

Figure 3.3: Comparison between obtained and expected outputs for increasing crossbar array sizes.

Although the slope of the obtained vs. expected conductance remains consistent across time steps, the thickness of the point clusters increases slightly. This broadening is primarily due to the accumulation of noise effects—such as read noise and conductance drift—which introduce variability in the measured values over time. However, the main effect that can be observed from the graphs in Figure 3.3 is a decrease in the expected values as the dimensions increase, meaning that the deviation from the ideal output becomes more pronounced. Indeed, the

parasitic resistance of the interconnects reduces the effective voltage across the cells and lowers the readout currents. As discussed in the previous chapter and highlighted by Equation (2.3), this phenomenon becomes increasingly severe as the number of bit lines grow, due to the longer conductive paths and the resulting cumulative voltage drops across the interconnects. Figure 3.3d confirms that the IR drop is the effective reason for these inaccuracies, demonstrating that removing its effect restores the linearity between the obtained and expected outputs.
The intensity of this effect also depends on the programmed conductance values. In particular, a high conductance corresponds to a small resistance; hence, for the same input voltage, the cell draws more current, which increases the voltage drop across the interconnects due to Ohm's law (V=IR). As a result, the IR drop becomes even more serious, further distorting the computation. In CMO-ReRAM crossbar arrays the effect is significant due to their inherently high maximum conductance, $G_{max} = 88\ \mu S$.
Figure 3.4 illustrates the evolution of the MVM error over time according to Equation 2.4 for different sizes: the vertical offset between the curves reflects the increasing error with array dimension, confirming the growing impact of IR drop in large-scale ReRAM arrays.

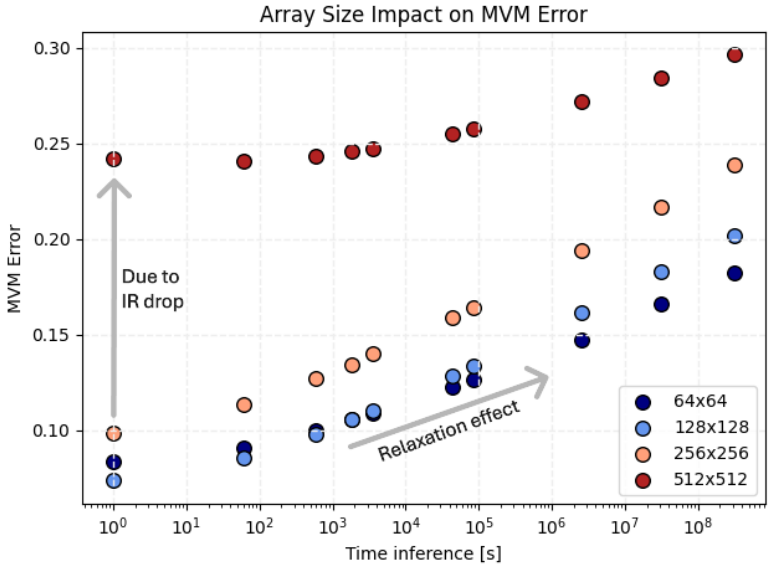
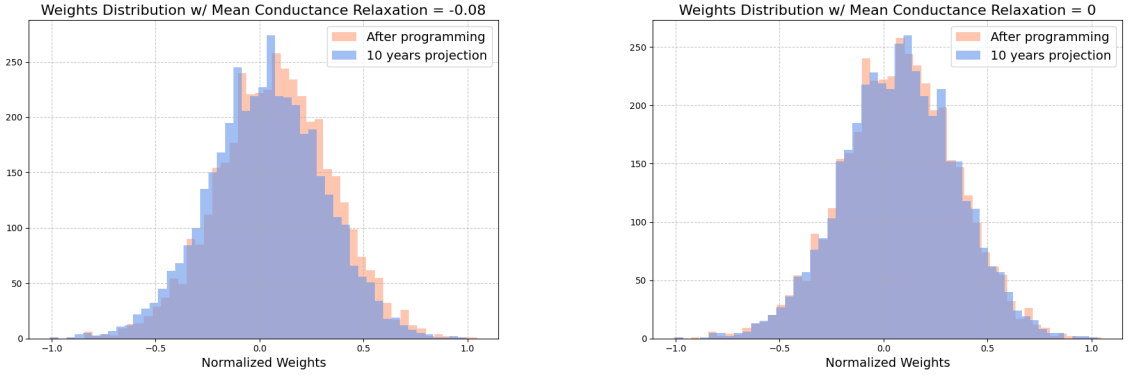


Figure 3.4: Evolution of MVM error over time for different crossbar array sizes.

In addition to this, each curve exhibits a noticeable slope, indicating that the error grows progressively with time. This time-dependent degradation is attributed

to the conductance relaxation, a critical issue that causes a gradual shift in the programmed states of ReRAM cells due to physical effects. As demonstrated by Falcone et al. in [8] and already mentioned in Section 2.1, the conductance relaxation is modeled as a Gaussian distribution whose mean decreases over time according to Equation 2.1, regardless of the conductance level. This behavior is confirmed in Figure 3.5a, where the weight distribution exhibits a shift towards high resistive states (lower conductances) over time, compared to the distribution observed immediately after programming. Although individual weight errors may appear minor, they can systematically reduce the MVM output. The impact becomes significant in large-scale neural networks, where errors accumulate across multiple layers, ultimately leading to a considerable drop in accuracy. Therefore, given a conductance relaxation effect characterized by a non-zero mean that is independent of the conductance levels, several strategies can be considered to compensate for this phenomenon, either in the digital or analog domain.



(a) (b)

Figure 3.5: Weight distribution immediately after programming and on a 10-year projection with: (a) experimentally observed mean of the conductance relaxation; (b) mean of the conductance relaxation set to 0.

The same analysis is repeated by changing the coefficient in Equation 2.1, whose experimentally observed value is -0.08, to 0: as shown in Figure 3.5b, the resulting weight distributions remain centered at both time instants. Although the weights still vary due to the increasing standard deviation over time, this variation follows a zero-mean Gaussian noise process, resulting in no systematic shift of the distribution and leading to some induced errors with opposite signs to cancel. This suggests that a possible mitigation strategy could involve implementing a

compensation mechanism, either in the analog or digital domain, to correct for the systematic current reduction introduced by the non-zero mean drift — specifically, the deviation of the actual current ($G_{\text{drift}} \cdot V$) from the ideal value ($G_{\text{prog}} \cdot V$). The heatmap in Figure 3.6 shows the MVM error as a function of both the crossbar array size and the mean of the conductance relaxation.

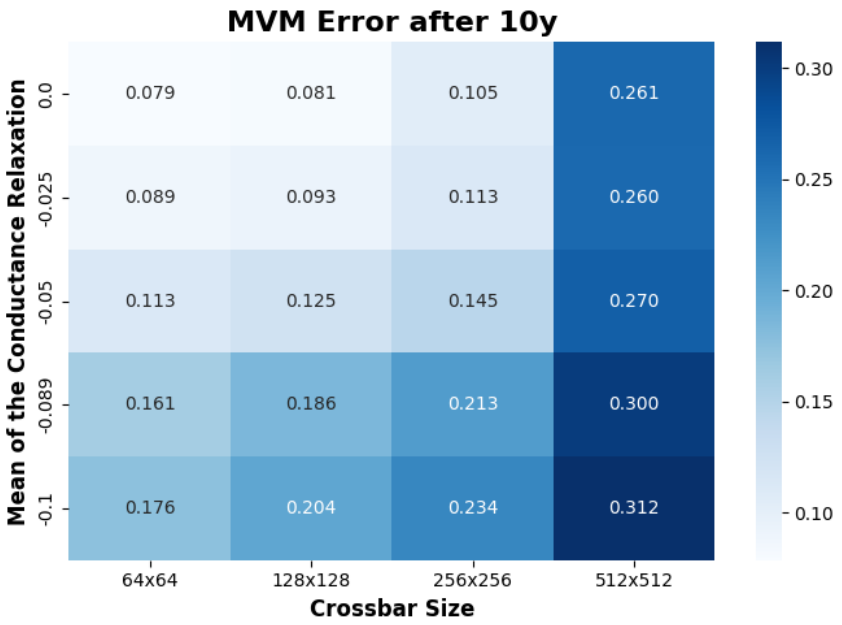


Figure 3.6: MVM error heatmap versus crossbar size and conductance relaxation mean coefficient.

The results highlight how higher values of the drift coefficient μ—which corresponds to stronger non-zero mean relaxation—lead to larger MVM errors over time. Since the mean of the conductance relaxation follows a $-\mu \cdot \log(t_{inf})$ trend (see Equation 2.1), minimizing μ would result in slower degradation and better retention of weight values. Although this coefficient represents an intrinsic property of the device and cannot be tuned arbitrarily, technologies that exhibit smaller drift coefficients could offer improved temporal stability. In addition to that, $log(t_{inf})$ is a known value, that can be anticipated during inference for fine-grained compensation.

The heatmap also shows that increasing the dimensions of the array results in higher MVM error due to amplified IR drop effects. In this case, a maximum array size of 256x256 could be a good compromise to limit this phenomenon, thus minimizing the overall error.

3.3 Neural Network Performance Over Time

The final part of this analysis focuses on the evaluation of neural network inference performances. Specifically, the previously introduced architectures—MLP, LeNet-5, and ResNet-32—are tested using analog ReRAM crossbar arrays, to assess their accuracy over time, considering five time stamps: immediately after programming, after $1\,s$, $1\,h$, $1\,day$ and $10\,years$. This temporal analysis is motivated by the conductance relaxation effect highlighted in the previous section, where the evolution of weight distributions over time clearly showed a shift towards lower values (see Figure 3.5). Although this degradation results in an increased MVM error, it does not necessarily translate into an immediate or proportional drop in inference accuracy. Indeed, due to the presence of activation functions and the inherent redundancy of deep neural networks, small deviations in individual matrix-vector multiplications can be partially compensated throughout the layers. Moreover, since classification relies on selecting the class with the highest output value, small distortions in the output vector do not necessarily affect the final prediction, as long as the correct class remains the same.

To account for IR drop effects, all architectures are implemented using a maximum crossbar size of 256x256, as determined in the previous paragraph in order to maintain error within an acceptable range. This constraint is especially relevant for the input and early convolutional layers, which often involve large matrix dimensions.

Hardware-aware trained models are also evaluated under the same conditions to enable a direct comparison.

Details regarding the structures of the neural networks under test and the inference setup can be found in Section 2.3.

3.3.1 MLP

The MLP model, previously introduced in Section 2.3.1, is a simple fully connected network composed of three layers and trained on the MNIST dataset. Its straightforward architecture makes it a suitable baseline to assess the impact of hardware non-idealities on inference accuracy. The translation into analog hardware can be

easily realized as shown in picture 3.7. Due to the limits on the maximum crossbar size, the input layer is mapped onto three 256x256 crossbars.

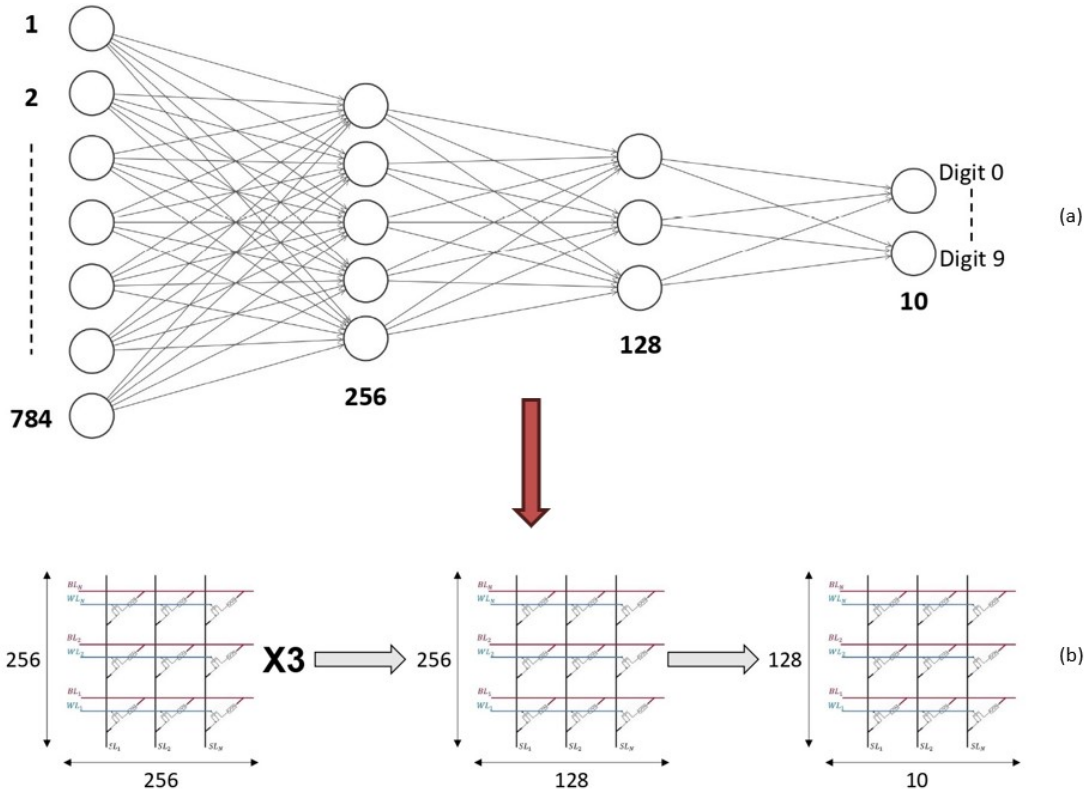


Figure 3.7: Hardware implementation of the MLP network using crossbar arrays.

To evaluate the effect of conductance drift over time, inference is performed at different time intervals after programming. The resulting classification accuracy is reported in Figure 3.8. Despite the limited number of linear layers, the impact of conductance relaxation becomes evident over longer time scales, causing a significant reduction in the accuracy after 10 years, whose value drops to 48.1%. However, performing the same simulation only considering a white Gaussian noise (i.e. zero mean), the accuracy remains nearly constant, demonstrating the critical role played by the conductance relaxation effect.

The inference accuracy of the MLP analog model is then compared to the one of the model after HWA Training (Figure 3.9). While the baseline accuracy progressively degrades due to the cumulative impact of hardware non-idealities, the HWA-trained model exhibits improved robustness, maintaining significantly higher accuracy over extended periods. This demonstrates the effectiveness of hardware-aware training in compensating for hardware non-idealities.

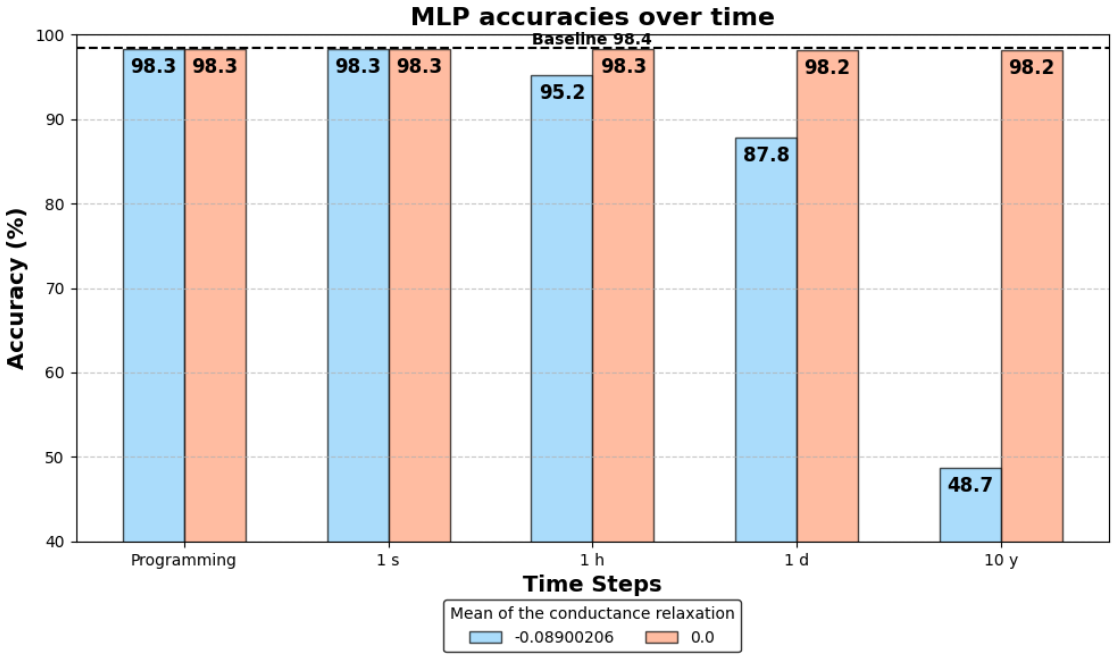


Figure 3.8: MLP accuracy evolution over time.

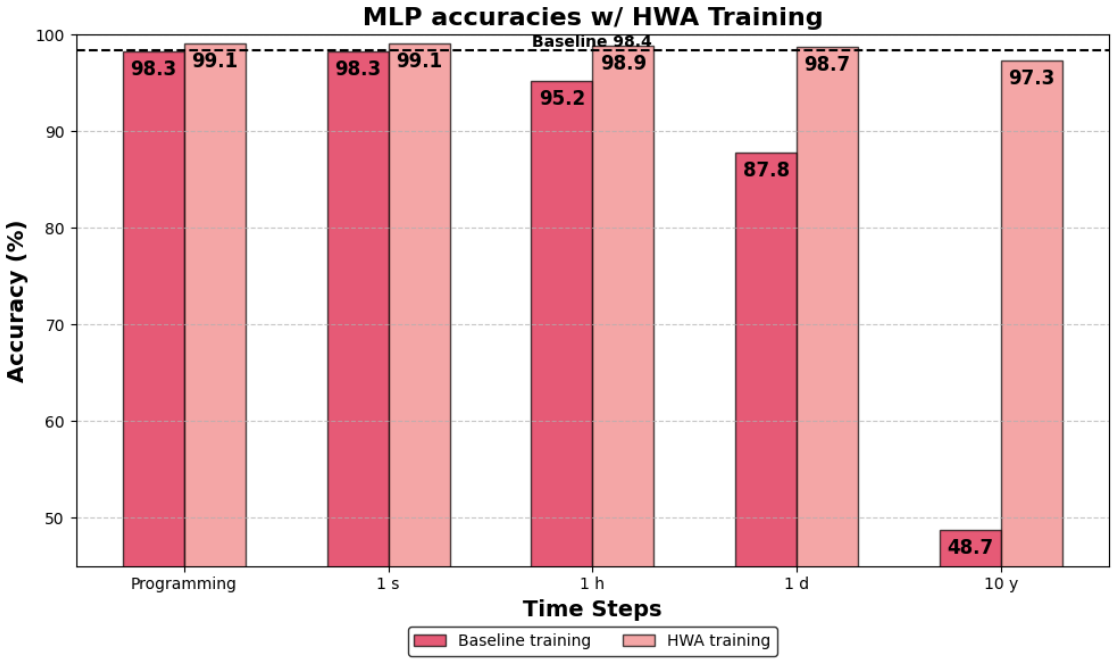


Figure 3.9: Performance comparison between baseline and hardware-aware trained models of MLP.

3.3.2 Lenet-5

LeNet-5 is a convolutional neural network architecture designed for image classification tasks. Compared to simpler models such as MLPs, it is capable of capturing more complex and spatially localized patterns from the input thanks to its convolutional filters and shared weights.
The pre-trained LeNet-5 model has been detailed in the previous chapter and is described in Table 2.2.

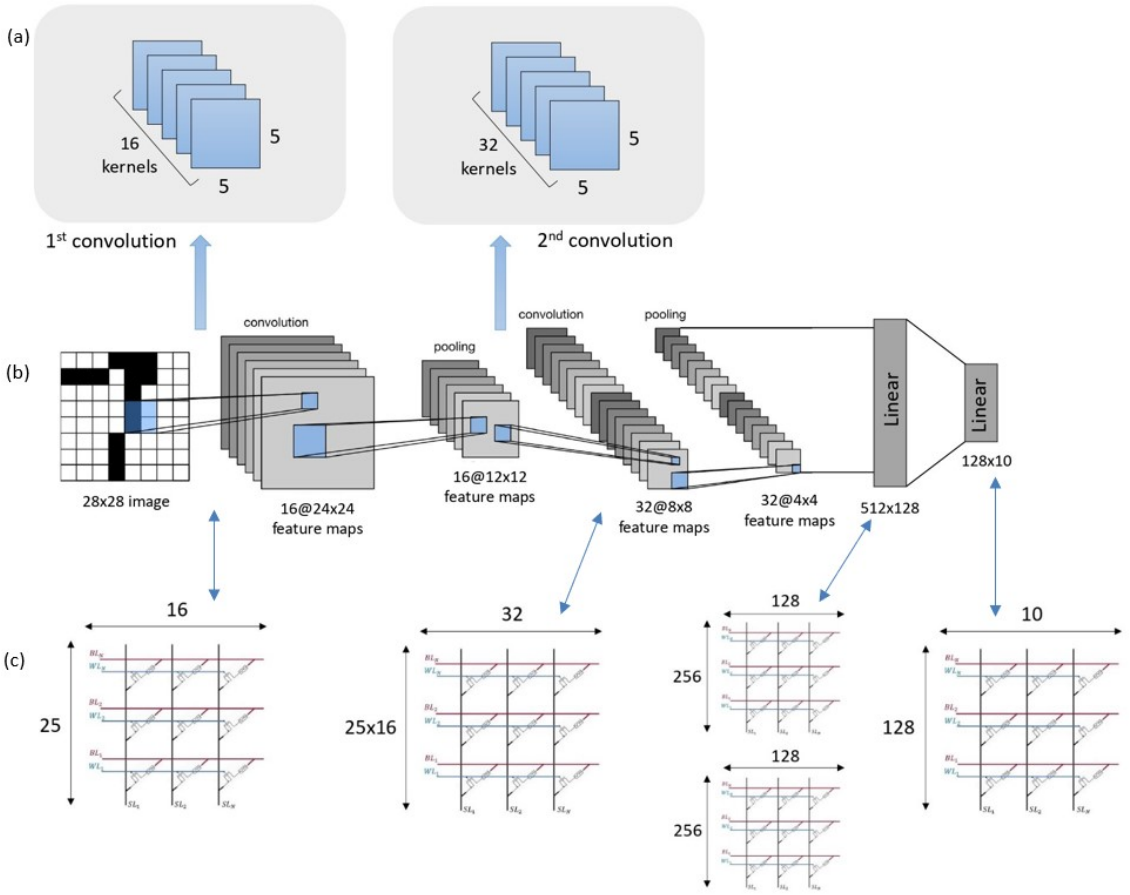


Figure 3.10: (a) Visualization of the filters in the two convolutional layers of LeNet-5: 16 filters of size 5×5 and 32 filters of size 5×5. (b) Full LeNet-5 architecture. (c) Mapping of LeNet-5 architecture onto an analog ReRAM crossbar array.

In order to transfer this architecture into hardware, weight matrices have to be mapped into a two-dimensional crossbar array. For convolutional layers, this operation is done by converting the filters into equivalent matrix forms compatible with the crossbar structure. Therefore, the first convolutional operation, which is conducted by means of 16 filters of size 5x5, is realized with a 16x25 crossbar ar-

ray. In the same way, the second convolution is performed with a 32x400 crossbar array, where 400 = 25x16. The whole structure is shown in Figure 3.10.
The accuracy over time is then analyzed to evaluate the network's robustness to temporal non-idealities such as conductance relaxation.
Also in this case, a progressive reduction in inference accuracy is observed as time advances, with an accuracy level equal to 71.8% after 10 years. This degradation is consistent with the conductance relaxation phenomenon previously analyzed. Indeed, repeating the simulations with zero mean conductance relaxation leads to a higher and more stable accuracy over time.

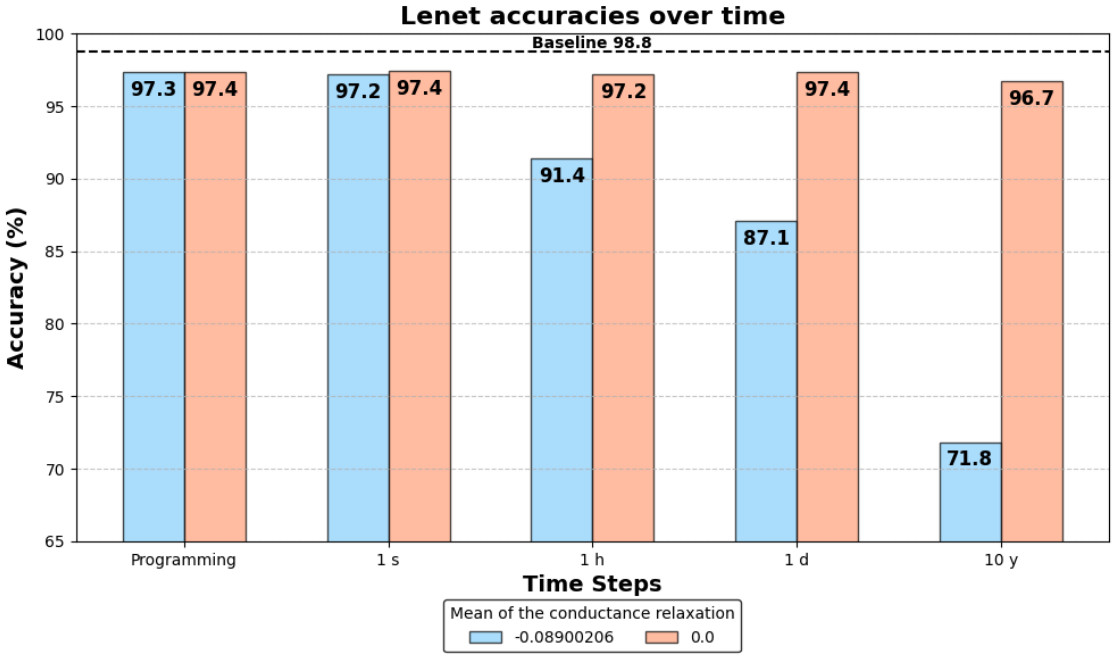


Figure 3.11: LeNet-5 accuracy evolution over time.

Although LeNet is architecturally more complex than an MLP structure, it shows better robustness to conductance relaxation over time. This is primarily due to the nature of convolutional layers, which differ significantly from fully connected layers in how they process information. In particular, three characteristics of convolutional networks should be discussed:

1. Convolutional layers use *local receptive fields*, which means that each neuron is connected only to a small region of the input image. As a result, even if a weight value changes, its effect is limited to a local area and does not affect the entire layer output, as it would in an MLP.

2. *Filters* (or kernels) in convolutional networks use the same set of weights across different regions of the input. Therefore, the total number of parameters is significantly reduced compared to a fully connected layer, and the cumulative impact of individual weight relaxation is limited.

3. Convolutional architectures perform *hierarchical feature extraction*. Early layers typically learn simple and general features, such as edges or corners, which are relatively robust to small changes in weights. These features are then combined in deeper layers to build more abstract representations. Even if some degradation occurs, the structure of this hierarchy helps maintain the essential information needed for classification.

In contrast, the MLP connects every input neuron to every neuron in the next layer, resulting in a much higher number of parameters and no weight sharing. This makes the MLP more vulnerable to cumulative drift effects, especially when the weights consistently decrease over time. As a result, the accuracy of the MLP degrades more significantly than LeNet over long time scales.

This temporal trend in accuracy is mirrored by the layer-wise error progression. Indeed, Figure 3.12, shows a vertical shift in the represented curves, meaning that the error progressively increases as time passes. Moreover, the analysis reveals an evident accumulation of the error as the input signal is processed through successive layers, reaching the highest peak at the last linear layer. However, this trend is sharply interrupted after the application of the `log-softmax` activation function. The reason is that `log-softmax` inherently compresses the output values into a normalized log-probability space, which tends to reduce the relative scale of the accumulated error. As a result, the discrepancy between the drifted and ideal outputs is significantly attenuated, especially in classification tasks where only the highest probability matters.

When the mean of the conductance relaxation is set to zero, as shown in Figure 3.13, the error across all layers is significantly lower, confirming that the non-zero mean drift is the main contributor to the observed degradation. This is further supported by the data in Figure 3.14, which shows how the layer-wise error varies with different values of the drift mean coefficient. Even small changes in this value can lead to non-negligible differences, underlining the importance of controlling

this effect to preserve accuracy over time.

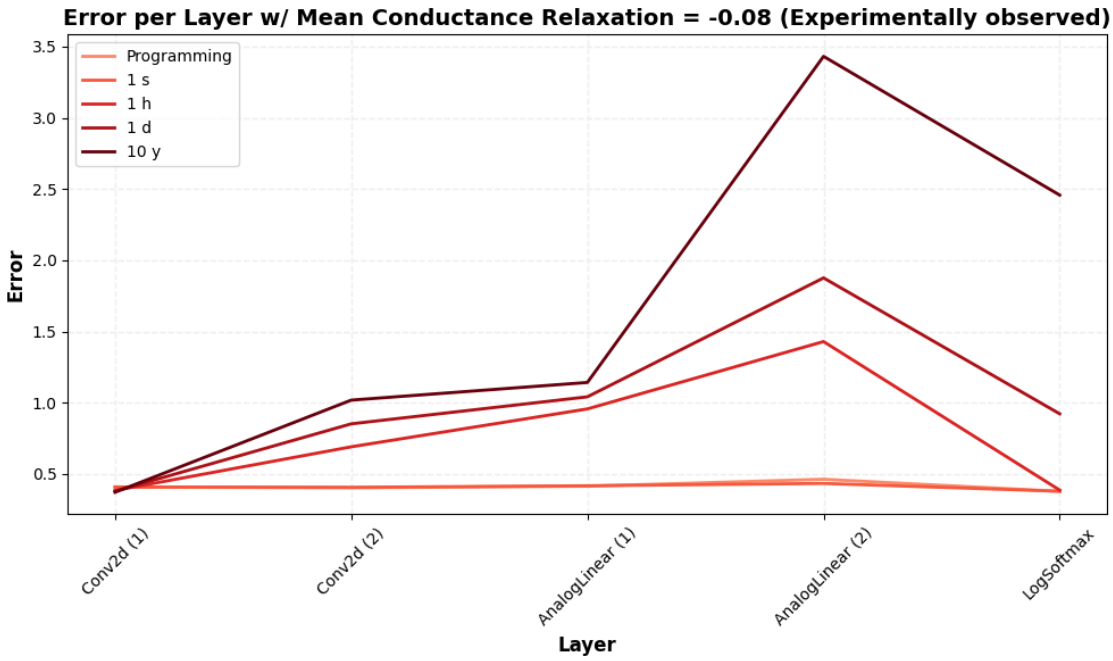


Figure 3.12: Error per layer over time with experimentally observed mean conductance relaxation.

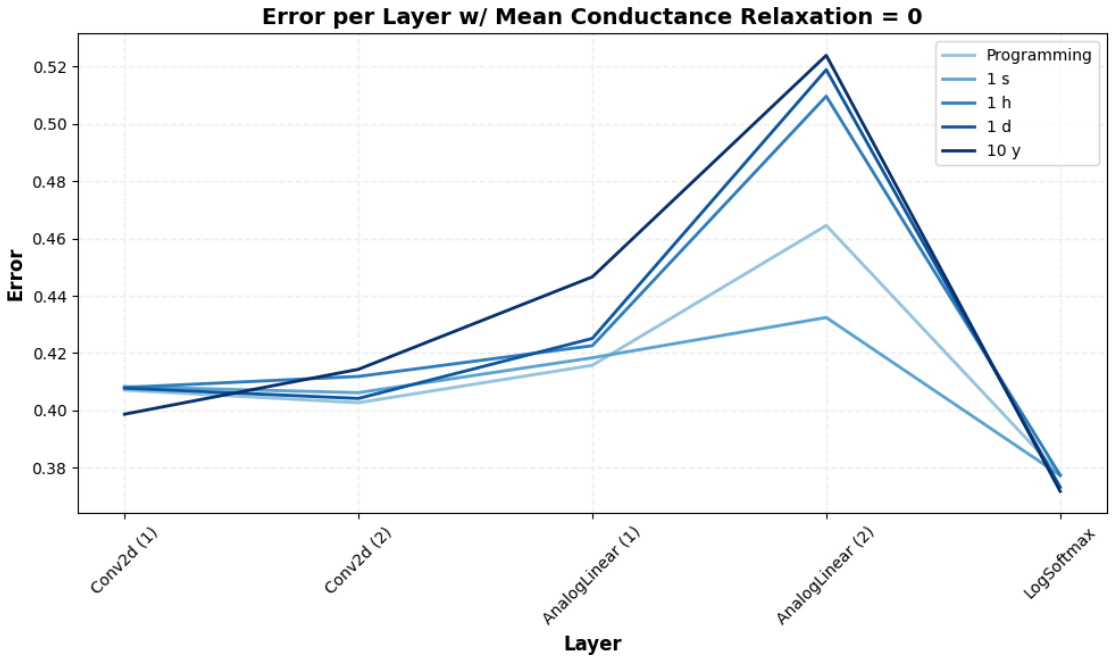


Figure 3.13: Error per layer over time with mean of the conductance relaxation set to 0.

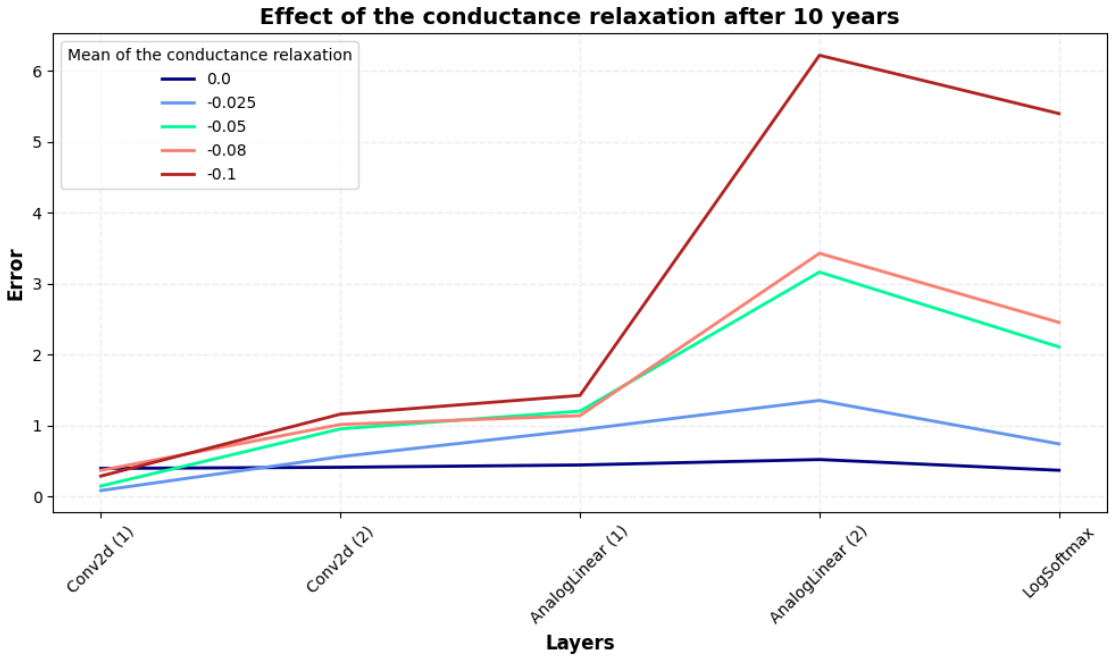


Figure 3.14: Error per layer after 10 years for different values of the conductance relaxation mean coefficient.

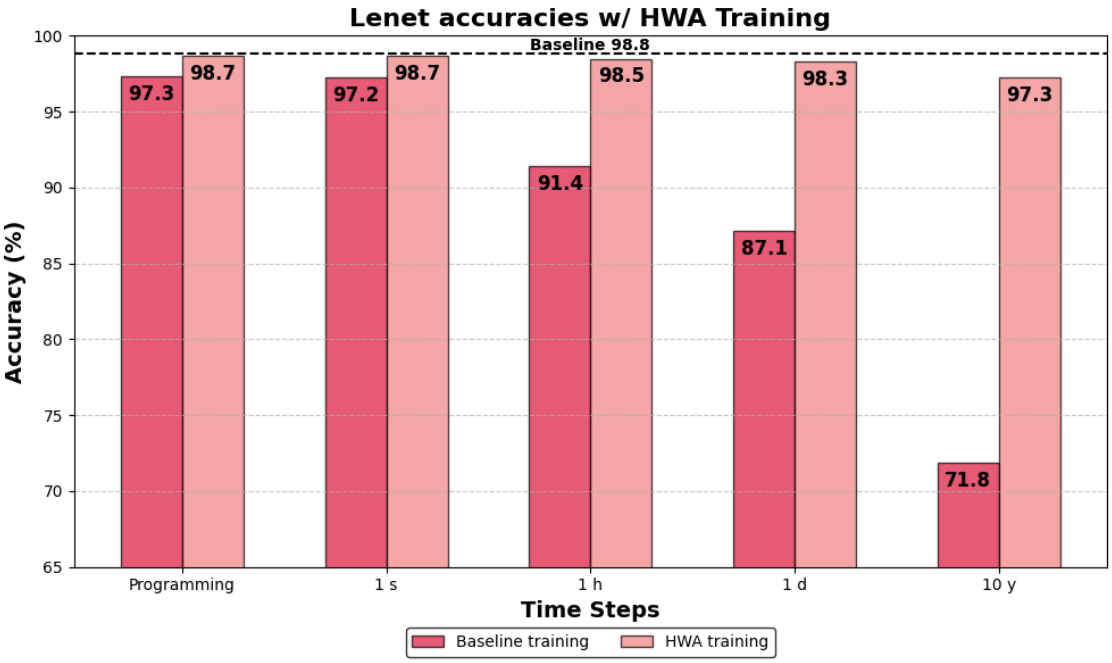


Figure 3.15: Performance comparison between baseline and hardware-aware trained models of LeNet-5.

Finally, the results obtained with the baseline model are compared with those of the model trained exploiting hardware-aware strategy. As expected, HWA training leads to a clear improvement in long-term inference accuracy, since the

model is trained taking into account hardware non-idealities and can, therefore, internalize these effects demonstrating greater robustness over time when performing inference (see Section 2.3.4 for details).

3.3.3 ResNet

To extrapolate to even more intricate architectures and complex tasks, the ResNet-32 model (Table 2.3) is adopted for the final simulations. Also in this case, the network architecture has to be mapped into hardware. Since the structure consists of convolutional layers, the analog simulator is realized in the same way as LeNet-5, e.g. the first convolutional layer with 16 filters of size 3x3 is transferred into a 16x9 crossbar array.

In this case, due to the deeper structure and residual nature of the network, the impact of conductance relaxation is even more pronounced, resulting in significantly lower performance over time. In particular, the inference accuracy drops to 10.5% already after 1 hour, while by mitigating the effects of the non-zero mean drift, either in digital or in analog, the accuracy shows high values even after 10 years, as illustrated in the figure below.

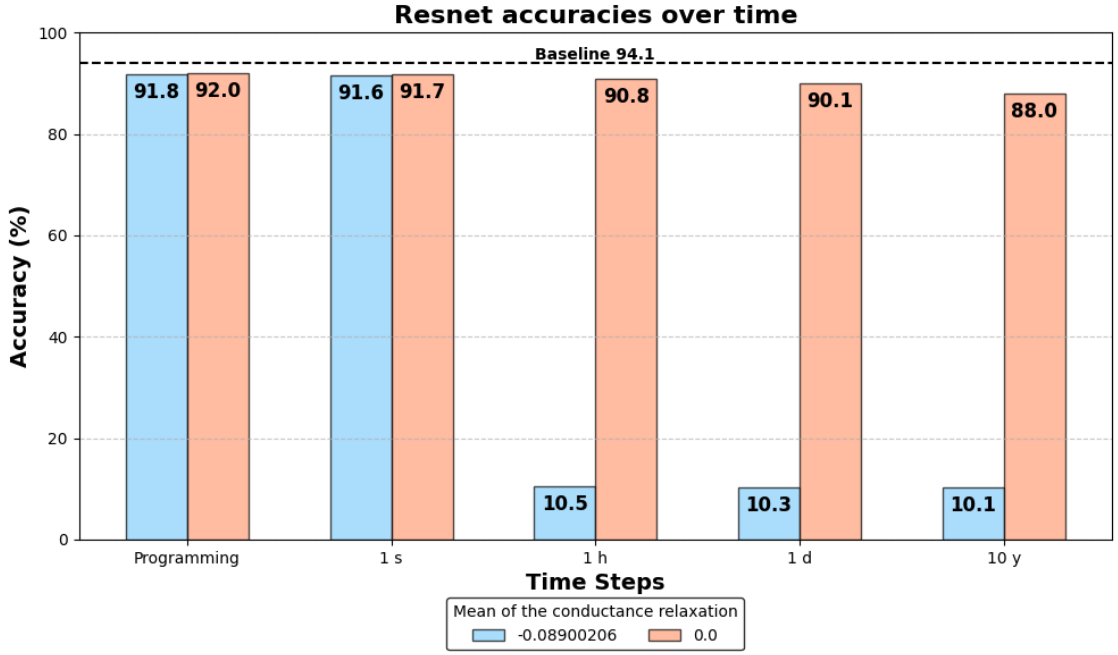


Figure 3.16: Accuracy over time for ResNet network.

As a matter of fact, residual neural networks are designed to mitigate the vanishing gradient problem throughout the adoption of skip (or residual) connections (Figure 2.9). Although this strategy enables a more effective learning, the additive nature of residual connections causes critical amplification of small errors over time: the output of a residual block is added directly to the original input. As a result, any error in the block's output is effectively reinforced and propagated through the network, leading to a cascading accumulation of inaccuracies that degrades the model's performance over time.

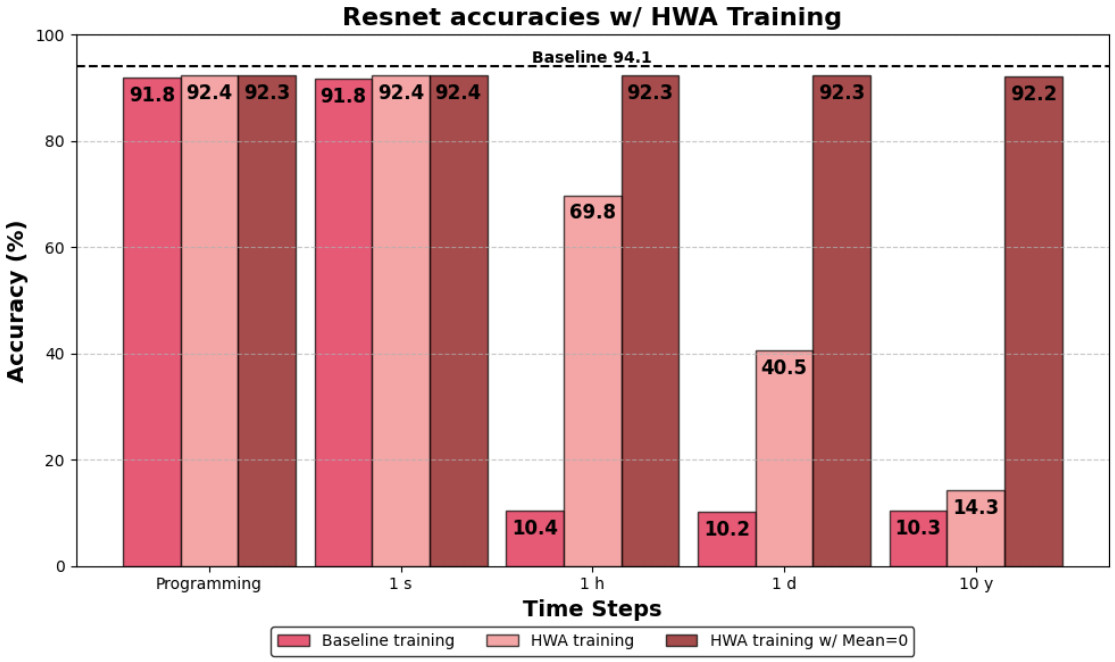


Figure 3.17: ResNet accuracies over time comparing three different approaches: baseline model, HWA trained model, HWA trained model disabling the effect of conductance relaxation.

Figure 3.17 shows that even with hardware-aware training, the robustness of the model is limited and no clear benefits are observed in the long term, despite an improvement of approximately 60% at $t_{inf} = 1\ h$. The main reason is that this strategy aims to account for analog non-idealities during training in order to make the network more resilient to them; however, it does not model the conductance relaxation effect, which is a long-term time-dependent phenomenon and, therefore, more critical to address.

In the case of LeNet-5, hardware-aware training alone is sufficient to ensure high accuracy even in the long term. This can be attributed to the simplicity of the ar-

chitecture, which makes it more robust to weight perturbations, and to the fact that HWA training improves resilience to variability by injecting additional white noise during training. Although this noise does not capture the systematic shift caused by the non-zero mean drift, it partially accounts for the standard deviation of the conductance relaxation and helps the network generalize well.

In contrast, for ResNet-32, a third simulation performed with hardware-aware trained model and considering the mean of the conductance relaxation equal to 0, validates the severity of this effect, and highlights how the combination of both strategies could be key to achieving high and stable accuracy over time, leading to more robust and reliable neural networks.

4 Conclusions

This thesis addressed the use of CMO-HfO_x ReRAM devices for neural network inference in AIMC systems, focusing on realistic hardware-aware simulations that incorporate both device-level non-idealities and crossbar-level parasitic effects.
The first part of the work involved the development and experimental validation of a statistical model for read noise, which was subsequently integrated with existing models for programming noise and conductance relaxation. This comprehensive modeling framework was implemented within the IBM Analog Hardware Acceleration Kit to enable a detailed and realistic simulation of inference processes.
Two types of simulations were performed:

- Crossbar array level simulations were conducted to evaluate the degradation in conductance accuracy caused by intrinsic noise and IR drop in memristive arrays.

- Neural network simulations were carried out to assess the effect of device non-idealities on inference accuracy over time, using three different architectures — MLP, LeNet-5, and ResNet-32 — applied to different tasks.

The study also explored the effectiveness of Hardware-Aware Training as a mitigation strategy to reduce inference degradation caused by time-dependent variations in device conductance.
The statistical model for the read noise was derived based on the electrical characterization presented in [19]. The experimental data revealed a logarithmic dependence of the noise standard deviation on the inference time and conductance level. These dependencies were captured by fitting the measured data within the conductance range of interest and were subsequently used to construct a realistic noise model. The model was then integrated with existing models for programming noise and conductance relaxation.
Crossbar array level simulations highlight the impact of non-idealities on the accuracy of conductance representation, considering the parasitic resistance of the wires. Voltage drops resulting from parasitic resistance along the bit lines de-

crease the effective voltage across ReRAM cells, leading to reduced readout currents. The severity of this effect increases with the dimensions of the array, due to the extended conductive paths and the accumulated resistance of the interconnect network.

In the same context, a shift towards high resistive states over time is observed in the weight distributions, caused by the effect of conductance relaxation. This phenomenon becomes critical in large scale neural networks where the propagation of errors across layers progressively degrades the overall inference accuracy. The proposed architectures exhibit a significant drop in accuracy over extended time periods, while the adoption of a hypothetical compensation scheme may substantially mitigate this effect.

Finally, the adoption of HWA Training leads to a notable improvement in inference accuracy, as the training process incorporates the effects of hardware nonidealities such as programming error and IR drop. However, it is important to note that conductance relaxation is not considered during training, and its impact may still degrade performance over time.

Bibliography

[1] S Agatonovic-Kustrin and R Beresford. "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research". In: *Journal of Pharmaceutical and Biomedical Analysis* 22.5 (2000), pp. 717–727. ISSN: 0731-7085. DOI: https://doi.org/10.1016/S0731-7085(99)00272-1.

[2] Amirali Amirsoleimani et al. "In-Memory Vector-Matrix Multiplication in Monolithic Complementary Metal–Oxide–Semiconductor-Memristor Integrated Circuits: Design Choices, Challenges, and Perspectives". In: *Advanced Intelligent Systems* 2.11 (2020), p. 2000115. DOI: https://doi.org/10.1002/aisy.202000115.

[3] Mattia Boniardi et al. "Phase Change Memory: A Review on Electrical Behavior and Use in Analog In-Memory-Computing (A-IMC) Applications". In: *Advanced Electronic Materials* 10.12 (2024), p. 2400599. DOI: https://doi.org/10.1002/aelm.202400599.

[4] Li Chen et al. "Ferroelectric memory based on two-dimensional materials for neuromorphic computing". In: *Neuromorphic Computing and Engineering* 2.2 (Mar. 2022), p. 022001. DOI: 10.1088/2634-4386/ac57cb. URL: https://dx.doi.org/10.1088/2634-4386/ac57cb.

[5] Wooseok Choi et al. "Nonvolatile Resistive Memory Technology for Deep Neural Network Hardware Applications". In: *Non-Volatile Memory and Selector Devices: Technology and Applications* (2025).

[6] L. Chua. "Memristor-The missing circuit element". In: *IEEE Transactions on Circuit Theory* 18.5 (1971), pp. 507–519. DOI: 10.1109/TCT.1971.1083337.

[7] Leon O. Chua. "How we predicted the memristor". In: *Nature Electronics* 1 (2018), p. 322. DOI: https://doi.org/10.1038/s41928-018-0074-4.

[8] Donato Francesco Falcone et al. *All-in-One Analog AI Accelerator: On-Chip Training and Inference with Conductive-Metal-Oxide/HfOx ReRAM Devices*. 2025. URL: https://arxiv.org/abs/2502.04524.

[9] R. K.-A. Abu Sebastian Manuel Le Gallo and E. Eleftheriou. "Memory devices and applications for in-memory computing". In: *Nature Nanotechnology* 15 (2020), pp. 529–544. DOI: https://doi.org/10.1038/s41565-020-0655-z.

[10] GeeksforGeeks. *Neural Networks - A Beginner's Guide*. https://www.geeksforgeeks.org/machine-learning/neural-networks-a-beginners-guide/. 2023.

[11] Tayfun Gokmen, Malte J. Rasch, and Wilfried Haensch. "The marriage of training and inference for scaled deep learning analog hardware". In: *2019 IEEE International Electron Devices Meeting (IEDM)*. 2019, pp. 22.3.1–22.3.4. DOI: 10.1109/IEDM19573.2019.8993573.

[12] Peter Hess. *Why a decades old architecture decision is impeding the power of AI computing*. 2025. URL: https://research.ibm.com/blog/why-von-neumann-architecture-is-impeding-the-power-of-ai-computing.

[13] Daniele Ielmini and Giacomo Pedretti. "Device and Circuit Architectures for In-Memory Computing". In: *Advanced Intelligent Systems* 2.7 (2020), p. 2000040. DOI: https://doi.org/10.1002/aisy.202000040.

[14] In Hyuk Im, Seung Ju Kim, and Ho Won Jang. "Memristive devices for new computing paradigms". In: *Advanced Intelligent Systems* 2.11 (2020), p. 2000105.

[15] Seungchul Jung et al. "A crossbar array of magnetoresistive memory devices for in-memory computing". In: *Nature* 601.7892 (2022), pp. 211–216.

[16] R. Khaddam-Aljameh et al. "HERMES Core – A 14nm CMOS and PCM-based In-Memory Compute Core using an array of 300ps/LSB Linearized CCO-based ADCs and local digital processing". In: *2021 Symposium on VLSI Circuits*. 2021, pp. 1–2. DOI: 10.23919/VLSICircuits52068.2021.9492362.

[17] Manuel Le Gallo and Abu Sebastian. "An overview of phase-change memory device physics". In: *Journal of Physics D: Applied Physics* 53.21 (2020), p. 213002.

[18] Yann LeCun, Y. Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[19] Davide G. F. Lombardo et al. “Read Noise Analysis in Analog Conductive-Metal-Oxide/HfOx ReRAM Devices”. In: *2024 Device Research Conference (DRC)*. 2024, pp. 1–2. DOI: 10.1109/DRC61706.2024.10643760.

[20] Abhishek Nandy and Manisha Biswas. “Neural Network Basics”. In: *Neural Networks in Unity: C# Programming for Windows 10*. Berkeley, CA: Apress, 2018, pp. 1–26. ISBN: 978-1-4842-3673-4. DOI: 10.1007/978-1-4842-3673-4_1.

[21] Malte J. Rasch et al. “A Flexible and Fast PyTorch Toolkit for Simulating Training and Inference on Analog Crossbar Arrays”. In: 2021, pp. 1–4. DOI: 10.1109/AICAS51828.2021.9458494.

[22] Tommaso Stecconi et al. “Filamentary TaOx/HfO2 ReRAM Devices for Neural Networks Training with Analog In-Memory Computing”. In: *Advanced Electronic Materials* 8.10 (2022), p. 2200448. DOI: https://doi.org/10.1002/aelm.202200448. URL: https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/aelm.202200448.

[23] Neil C Thompson et al. “The computational limits of deep learning”. In: *arXiv preprint arXiv:2007.05558* 10 (2020).

[24] Hsinyu Tsai et al. “Recent progress in analog memory-based accelerators for deep learning”. In: *Journal of Physics D: Applied Physics* 51 (2018). URL: https://api.semanticscholar.org/CorpusID:125608390.

[25] Zhongrui Wang et al. “Resistive switching materials for information processing”. In: *Nature Reviews Materials* 5.3 (Mar. 2020), pp. 173–195. DOI: 10.1038/s41578-019-0159-3.

[26] H.-S. Philip Wong et al. “Metal–Oxide RRAM”. In: *Proceedings of the IEEE* 100.6 (2012), pp. 1951–1970. DOI: 10.1109/JPROC.2012.2190369.

[27] Wei Wu et al. “A Methodology to Improve Linearity of Analog RRAM for Neuromorphic Computing”. In: *2018 IEEE Symposium on VLSI Technology*. 2018, pp. 103–104. DOI: 10.1109/VLSIT.2018.8510690.

[28] Fan Zhang et al. “XMA: a crossbar-aware multi-task adaption framework via shift-based mask learning method”. In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, pp. 271–276. ISBN: 9781450391429. DOI: 10.1145/3489517.3530458.