LEARNING AND INFERENCE FOR WIRELESS COMMUNICATIONS APPLICATIONS USING IN-MEMORY ANALOG COMPUTING

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Muhammad Atif Ali

July 2024

Learning and Inference for Wireless Communications Applications using In-Memory Analog Computing
By Muhammad Atif Ali
July 2024

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Tolga Mete Duman (Advisor)

Sinan Gezici

Javad Haghighat

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

LEARNING AND INFERENCE FOR WIRELESS COMMUNICATIONS APPLICATIONS USING IN-MEMORY ANALOG COMPUTING

Muhammad Atif Ali
M.S. in Electrical and Electronics Engineering
Advisor: Tolga Mete Duman
July 2024

The exponential growth of wireless communication technologies has created a crucial need for more efficient and intelligent signal processing in decentralized devices and systems. Traditional digital computing architectures increasingly struggle to meet these rising computational demands, leading to performance bottlenecks and energy inefficiencies. The problem becomes more significant on edge devices with limited computing capabilities and severe energy limitations. Integrating machine learning algorithms with in-memory analog computing, specifically memristor-based architectures, provides a non-traditional computing paradigm and can potentially enhance the energy efficiency of edge devices. By leveraging the properties of memristors, which can perform both storage and computation, this research investigates ways to potentially reduce latency and power consumption in signal-processing tasks for wireless communications.

This study examines memristor-based analog computing for deep learning and inference in three areas of (wireless) communications: cellular network traffic prediction, multi-sensor over-the-air inference for internet-of-things devices, and neural successive cancellation decoding for polar codes. The research includes the development of robust training techniques for memristive neural networks to cater for degraded performance due to noise in analog computations and offer acceptable prediction accuracy with reduced computational overhead for network traffic management. It explores in-memory computing for an L_p-norm inspired sensor fusion method with analog sensors and enables more efficient multi-sensor data fusion. Also, it investigates the incorporation of analog memristive computing in neural successive cancellation decoders for polar codes, which could lead to more energy-efficient decoding algorithms. The findings of the thesis suggest

potential improvements in energy efficiency and provide insights into the benefits and limitations of using in-memory computing for wireless communication applications.

Keywords: analog computing, memristors, neural networks, deep learning, wireless communications, in-memory computing, cellular network traffic prediction, polar codes, neural successive cancellation.

ÖZET

BELLEK İÇİ ANALOG HESAPLAMA KULLANARAK KABLOSUZ İLETİŞİM UYGULAMALARI İÇİN ÖĞRENME VE ÇIKARIM

Muhammad Atif Ali
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Tolga Mete Duman
Temmuz 2024

Kablosuz iletişim teknolojilerindeki üstel büyüme, merkezi olmayan cihazlar ve sistemlerde daha verimli ve akıllı sinyal işleme ihtiyacını doğurmuştur. Geleneksel dijital hesaplama mimarileri, artan bu hesaplama taleplerini karşılamada giderek zorlanmakta, bu da performans darboğazlarına ve enerji verimsizliklerine yol açmaktadır. Bu sorun, sınırlı hesaplama yeteneklerine ve ciddi enerji sınırlamalarına sahip uç cihazlarda daha da önemli hale gelmektedir. Makine öğrenimi algoritmalarının bellek içi analog hesaplama ile, özellikle memristör tabanlı mimarilerle entegrasyonu, geleneksel olmayan bir hesaplama paradigması sunmakta ve uç cihazların enerji verimliliğini potansiyel olarak artırabilmektedir. Bu araştırma hem depolama hem de hesaplama yapabilen memristörlerin özelliklerinden faydalanarak, kablosuz iletişimin sinyal işleme görevlerinde gecikme ve güç tüketimini potansiyel olarak azaltma yollarını incelemektedir.

Bu çalışma, kablosuz iletişimde derin öğrenme ve çıkarım için memristör tabanlı analog hesaplamayı üç ana alanda incelemektedir: hücresel ağ trafiği tahmini, nesnelerin interneti cihazları için çoklu sensörlü hava üstü çıkarım ve kutup kodları için sinirsel ardışık iptal kod çözme. Araştırma, ağ trafiği yönetimi için azaltılmış hesaplama yükü ile kabul edilebilir tahmin doğruluğu sunan analog hesaplamalardaki gürültü nedeniyle performansın düşmesine karşı dayanıklı eğitim tekniklerinin geliştirilmesini içermektedir. Ayrıca, analog sensörler ile bellek içi hesaplama kullanılarak L_p-norm ilhamlı bir sensör füzyon yöntemi için daha verimli çoklu sensör veri füzyonunu araştırmaktadır. Bunun yanı sıra, kutup

kodları için sinirsel ardışık iptal kod çözücülerine analog memristör hesaplamasının entegrasyonunu inceleyerek, daha enerji verimli kod çözme algoritmalarına ulaşılabileceğini öne sürmektedir. Tezin bulguları, enerji verimliliğinde potansiyel iyileştirmeler önermekte ve kablosuz iletişim uygulamaları için bellek içi hesaplamanın faydalarını ve sınırlamalarını anlamamızı sağlamaktadır.

Anahtar sözcükler: Analog hesaplama, memristörler, sinir ağları, derin öğrenme, kablosuz iletişim, bellek içi hesaplama, hücresel ağ trafiği tahmini, polar kodlar, sinirsel ardışık iptal.

Acknowledgement

I express my deepest gratitude to my advisor, Prof. Tolga Mete Duman. This journey would have been an impossible challenge without his unwavering guidance and support. His immense knowledge, dedicated help, and infinite patience have made the impossible achievable and truly rewarding.

I want to thank Prof. Sinan Gezici and Assoc. Prof. Javad Haghighat for their time, perceptive comments and feedback. It has significantly enhanced the quality of my work.

I want to express my gratitude for the financial support received from TÜBİTAK through the CHIST-ERA project SONATA (CHIST-ERA-20-SICT-004), funded by TÜBİTAK, Grant 221N366.

The success of my research was only possible due to the collaborative and inspiring atmosphere of the Bilkent Communication Theory and Application Research (CTAR) Lab. I extend my sincere appreciation to all the lab members: Mert Özateş, Mohammad Javad Ahmadi, Furkan Bağcı, Uras Kargı, and Dr. Mohammad Kazemi. I also acknowledge and express my gratitude to Dr. Büşra Tegin, my valued research collaborator and mentor on this journey. Her constant guidance propelled this work to completion.

I cannot express enough my gratitude to my family for their endless and unconditional support throughout this journey. My wife, Hina Manzoor, has been my infinite source of motivation during every low point, always supporting me when I falter. My son, Muhammad Zumar, thank you for letting me forget work stress and enjoy our beautiful moments together.

This journey would not have been possible without these individuals' collective support and encouragement. I am genuinely grateful for their contributions to my academic and personal growth.

Contents

List of Figures

List of Tables

Acronyms

1T1R	one-transistor one-memristor
ADC	analog-to-digital converter
AI	artificial intelligence
ANN	artificial neural network
AWGN	additive white Gaussian noise
BER	bit error rate
BS	base station
CBRAM	conductive bridge random access memory
CMOS	complementary metal oxide semiconductor
CNN	convolutional neural network
CPU	central processing unit
CSI	channel state information
DAC	digital-to-analog converter
DL	deep leaning
DNN	deep neural network
DRAM	dynamic random access memory
EcRAM	electrochemical random access memory
GPU	graphical processing unit
HP	Hewlett-Packard
IMC	in-memory computing
IoT	internet of things
KVL	Kirchhoff's voltage law
LLR	log-likelihood ratio

LTE	long-term evolution
MAC	multiple access channel
MAE	mean absolute error
ML	machine learning
MLP	multi layer perceptron
MSE	mean squared error
MVM	matrix vector multiplication
NN	neural network
NSC	neural succesive cancellation
OTA	over-the-air
PCM	phase change memory
PDCCH	physical downlink control channel
QoS	quality-of-service
ReLU	rectified linear unit
ReRAM	resistive random access memory
SC	succesive cancellation
SNR	signal-to-noise ratio
SRAM	static random access memory
TEAM	threshold adaptive memristor model
VTEAM	voltage-controlled threshold adaptive memristor model

Chapter 1

Introduction

Machine learning (ML) is now a universal tool actively used in almost all aspects of life. It offers accessible and efficient solutions to complex problems, such as image recognition and classification, recommendation systems, speech synthesis and recognition, machine translation, and generative artificial intelligence (AI) i.e., large language models, etc. ML solutions often employ neural networks (NNs) that select the most suitable outcomes based on probabilistic models. On a different front, the rapid growth of wireless communication technologies has significantly increased the demand for efficient and intelligent signal processing capabilities at the edge of wireless networks [1]. While ML can significantly improve the performance of solutions to many complex problems, deploying them on edge devices poses challenges due to computational demands and power consumption. These devices are often energy-constrained, with limited battery life and computing capabilities. Another issue affecting performance is inference latency, which limits the use of these solutions in real-time applications. Some approaches focus on compression and pruning techniques to make ML solutions viable on edge devices, but these often result in a loss of performance [2]. To address these challenges, combining ML solutions with in-memory analog computing emerges as a promising solution to improve the practicality and efficiency of deep learning systems.

In-memory analog computing leverages the physical properties of electronic devices to perform computations directly within memory, enabling highly parallel and energy-efficient data processing. This approach contrasts with conventional digital computing, which is often limited by the von Neumann bottleneck, where data transfer between the processor and memory limits the overall system performance. A trending device in advancing in-memory computing technologies is the memristor, first described by Professor Leon O. Chua 50 years ago [3] and later realized in a prototype device by HP Labs in 2008 [4]. The conductance of a memristor can be tuned by applying voltage pulses of varying amplitude, polarity, and duration. This characteristic, combined with their high compatibility with complementary metal oxide semiconductor (CMOS) technology, compact size for high-density chip integration, non-volatility, fast operation, low energy consumption, and excellent scalability, makes them an excellent candidate for in-memory computing.

Recent research has shown the feasibility of using memristors for storage [5] and in-memory analog computing [6]. In-memory computing based on memristors can provide energy efficiency up to 100 times greater than traditional central processing unit (CPU) and graphical processing unit (GPU) computations [7], presenting a significant advantage for machine learning applications in energy-constrained edge devices.

When organized into crossbar arrays, memristors can act as excellent neural network accelerators as the crossbar architecture can efficiently perform matrix vector multiplication (MVM), a fundamental operation in many machine learning algorithms. This makes studying the applications of in-memory analog computing using memristors an interesting use case. Considering the potential energy savings and reduced latency, it is worth exploring their use for applications in wireless communications.

1.1 Contributions of the Thesis

The contributions of this thesis are multifaceted, addressing multiple applications in wireless communications using in-memory computing. In Chapter 3, we demonstrate the efficacy of analog computations for network traffic prediction, specifically leveraging memristive devices. Our experimental framework, adapted from existing neural network architectures, introduces a robust training approach to improve performance loss due to noise in analog computations, enhancing robustness against non-idealities inherent to memristors.

We experimented with three distinct techniques for robust training by utilizing phase change memory (PCM) noise, multiplicative log-normal noise, and additive white Gaussian noise (AWGN) during forward pass computation. We provide comprehensive insights into their impact on model performance over extended periods. Our findings highlight that robust training with additive white Gaussian noise significantly mitigates the degradation in prediction accuracy due to conductance drift, thus providing the best performance for long-term deployment of analog neural networks in real-world applications.

In Chapter 4, our contributions focus on deploying deep neural networks (DNNs) on edge internet of things (IoT) devices, particularly in the context of multi-sensor data fusion. We build on the work of [8, 9], which introduces a learnable L_p-norm inspired sensor fusion method with over-the-air (OTA) aggregation. This method allows for an efficient fusion of data from multiple spatially distributed sensors, enabling the system to perform robust and accurate inference despite the inherent limitations of edge devices. Using a learnable parameter in the L_p-norm function ensures that the sensor fusion process is optimized for various environmental conditions and network characteristics, enhancing the system's adaptability and overall performance.

Our comprehensive experimental evaluation includes simulations of memristive behavior, accounting for programming noise, drift noise, and read noise, to accurately model the real-world performance of memristive neural networks. We

conduct extensive testing under different signal-to-noise ratio (SNR) conditions, demonstrating that our approach maintains high accuracy and robustness even in challenging scenarios. Our results highlight the substantial energy efficiency gains achieved through memristor-based in-memory computing, making this approach highly suitable for battery-powered and energy-constrained IoT applications.

Additionally, we explore the impact of drift noise over extended periods, providing valuable insights into the long-term reliability of memristive neural networks. Our findings suggest that while there is an initial drop in accuracy due to noise, the system remains robust and maintains acceptable performance levels over time. This research marks a significant advancement towards realizing the full potential of integrating DNNs with IoT, offering a robust, energy-efficient solution for complex computational tasks in distributed environments.

In Chapter 5, our contributions center around addressing the complexity and latency issues of a neural succesive cancellation (NSC) decoder for polar codes by introducing an equivalent analog NSC decoder that leverages memristors for analog in-memory computing. We replace all layers of the NSC decoder with equivalent analog layers, utilizing memristors to perform matrix-vector multiplications efficiently. We address the high computational and power costs of DNNs on edge devices and focus on low-latency and energy-efficient in-memory computing capabilities of the analog NSC.

We explore the impact of non-ideal characteristics of memristors, such as programming noise, drift noise, and read noise, on the performance of the analog NSC decoder. While there is a drop in performance as compared to digital computations due to the noisy nature of analog computations, this approach provides significant advantages over traditional computing architectures by reducing latency and energy consumption, making it highly suitable for real-time applications.

We present the bit error rate (BER) versus SNR for different configurations of the NSC decoder. Our findings indicate that the analog NSC decoder loses less than 0.5 dB when compared to the NSC decoder for the same BER. Additionally, we study the performance of the analog NSC decoder when programming and drift

noises affect the network weights, and observe that the BER gradually increases for the same SNR conditions as the drift time increases.

Our study highlights the practical benefits and challenges of deploying neural network-based decoders for polar codes using memristor technology, providing a robust framework for future innovations in energy-efficient, high-performance decoding algorithms.

1.2 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2, we provide a detailed overview of the preliminaries on memristors, their technologies, and simulation frameworks, along with an overview of the existing literature on these topics. Chapter 3 focuses on applications of memristive neural networks for real-time cellular network traffic and compares energy savings. Our study on multi-sensor inference with neural networks using memristor-based analog computing is presented in Chapter 4. Chapter 5 discusses the neural successive cancellation decoder for polar codes using analog in-memory computing. In Chapter 6, we present our conclusions and future research directions.

Chapter 2

Preliminaries and Literature Review

Deep neural networks (DNNs) excel at managing complicated tasks, enabling high accuracy and generalizable performance in areas such as image recognition and classification, natural language processing, predictive analytics, generative AI, and numerous other applications. As the difficulty of the tasks increases, so does the size and complexity of the networks, which demands increasingly energy-efficient and powerful processors. Hardware fine-tuned for DNNs, specifically for MVM operations, enables machine learning advancements for complex networks [10]. However, more efficient hardware is required to keep up with rapid advancements in the field. Integrating DNNs into IoT devices for edge intelligence requires a high computational and power cost. Furthermore, IoT devices' limited power and computing capability causes high latency and power draw when using DNNs for inference.

In-memory computing, which uses memory for both processing and storage, has garnered considerable interest as a potential solution offering substantial enhancements in energy efficiency [11, 12, 13]. These accelerators use the analog nature and non-volatile memory devices to perform MVM operations in a single

step. Despite the inherent advantages of analog processing, maintaining consistent accuracy is difficult. Unlike digital systems, the fidelity of solutions in analog processing is inherently uncertain and is directly influenced by electrical noise, process variations, and various other parasitic effects. To get precision similar to digital systems, many prior works adopted a hybrid approach called bit slicing, where the network weight values are split bitwise across multiple devices, digitized and aggregated [11, 12, 13]. This approach allows for the storage of weights with higher precision, even when using low-precision devices, though it incurs a higher energy cost than a purely analog method. Another analog methodology for storing network weights on noisy memory devices investigates adaptive redundancy, adaptive mapping, and sign protection utilizing the sensitivity and sparsity of the network [5].

Memristors are promising candidates for in-memory computing because they co-locate storage and computing by harnessing the non-volatility property of memory arrays. They can perform many concurrent multiply-and-accumulate operations in a single step, resulting in the following important benefits:

1. Low latency, as no data is being moved back and forth between the memory and processor. This enables faster processing of data, which is essential for real-time applications such as autonomous vehicles and robotics.

2. Energy efficiency, due to a unified non von Neumann architecture. This can significantly reduce the energy consumption of deep learning systems, making them more sustainable and eco-friendly

Memristors can solve both high latency and high energy consumption problem of DNNs through their inherent low-power in-memory computing ability. They provide a promising alternative to traditional computing hardware, such as CPUs and GPUs, which consume large amounts of energy and require high latency to process data. CPUs and GPUs are based on the von Neumann architecture, where computing and storage are physically separated. Data must be moved to and from the attached memory to the processor. This creates a bottleneck

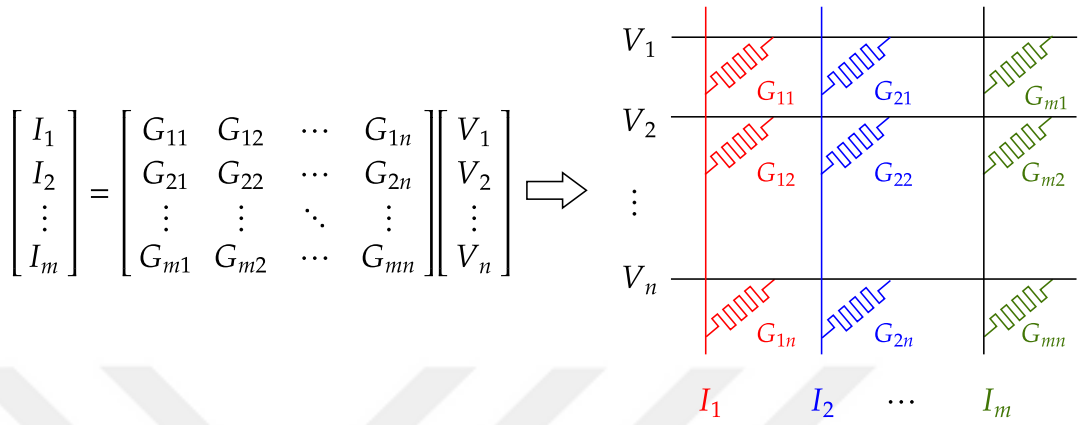

Figure 2.1: A matrix-vector multiplication on a memristive crossbar simulating memristive neural network layer.

and increases latency and energy consumption [7]. In classical hardware-based artificial neural networks (ANNs), fundamental operations such as multiplication, addition, and activation are carried out via CMOS circuits, particularly GPUs. The weight parameters are stored in static random access memory (SRAM) or dynamic random access memory (DRAM)-based memory, and computation is carried out by CPUs or GPUs. A physical division between these two subsystems requires continuous data transfer from memory to computational units. This issue becomes particularly pronounced as the network scales. For example, a two-layer 784-800-10 fully connected neural network for classifying MNIST digits [14] has 635,200 interconnections and a more complex model like ResNet-50, used for large-scale image classification tasks, has over 23 million parameters, resulting in significantly higher data transfer demands between the CPU and memory [15].

The MVM is the most fundamental mathematical operation in deep neural networks, and memristors arranged in a crossbar fashion can perform this very efficiently in a concurrent manner. Figure 2.1 shows a matrix-vector product performed on a memristor crossbar using the Kirchhoff's voltage law (KVL).

This chapter is structured to provide a comprehensive overview of DNNs, in-memory computing, and memristor technology. It begins with an introduction to DNN capabilities and challenges, emphasizing the need for energy-efficient hardware for MVM operations. In-memory computing is discussed as a solution for

enhancing energy efficiency, followed by exploring memristors' potential due to their low latency and energy efficiency. Section 2.1 details different types of memristor technologies. Next, a few existing hardware implementations are discussed in Section 2.2, showcasing the potential of memristor-based systems. Section 2.3 discusses various memristor models in the literature. The chapter explains the stochastic inference model for PCM-type memristors, focusing on noise sources and their impacts in Section 2.4. Section 2.5 briefly describes some existing memristive neural network simulation frameworks. The chapter is summarized in Section 2.6.

2.1 Memristor Technology

Memristors offer an exciting area of research that could significantly impact the field of deep learning specifically for energy constrained and latency dependent applications. They are electronic devices with a unique ability to remember the past and their resistance changes depending on the amount of current that has previously flowed through them. This property of memristors can be harnessed to create an in-memory computing architecture.

Figure 2.2 illustrates typical $I - V$ curve of a memristor. The curve exhibits a pinched hysteresis loop shape, which is distinct from the behavior of circuits composed solely of simple R, L, and C elements. This unique loop characteristics allows memristors to retain a memory of past voltage or current, making them highly beneficial for applications in non-volatile memory technologies and in-memory computing.

2.1.1 Types of Memristors

Memristors are based on different technologies, and several of them [16] have been proposed in the literature for deep learning applications.

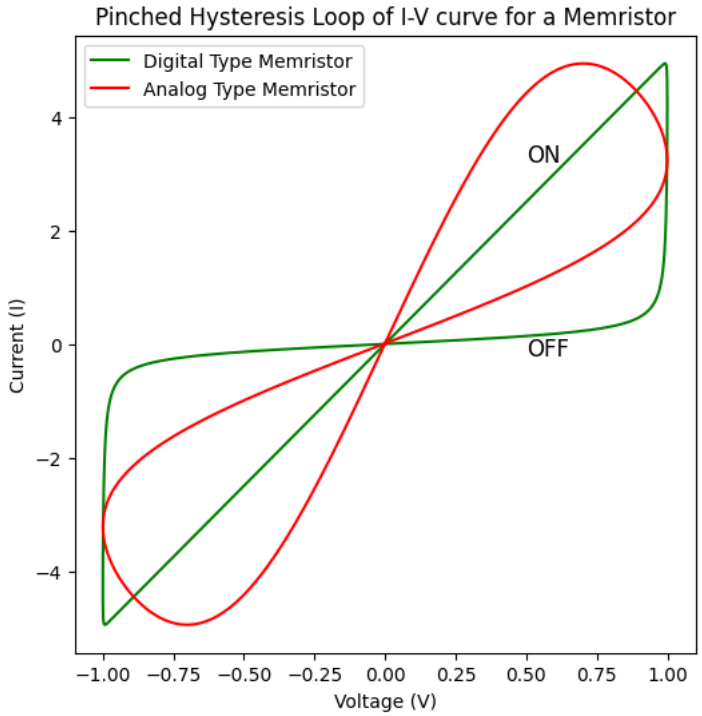

Figure 2.2: Pinched hysteresis $I - V$ loop of memristor.

2.1.1.1 Metal Oxide Based Memristors

Metal oxide-based memristors are characterized by resistive switching behavior, often modeled using the drift-diffusion model [17]. This model explains how the movement of oxygen vacancies under an applied electric field alters the device's resistance. When voltage is applied, these vacancies drift, causing changes in the resistance. This resistive switching is observed in many metal oxide systems, popularly in Ta_2O_5, HfO_2, and TiO_2 based devices.

2.1.1.2 Phase Change Memory (PCM)

A PCM memristor is a type of memristor that utilizes the phase transition of a material to switch between different states. These devices typically involve transitions between the amorphous and crystalline states in materials such as GeSbTe alloys. PCM memristors are known for their ability to represent information in

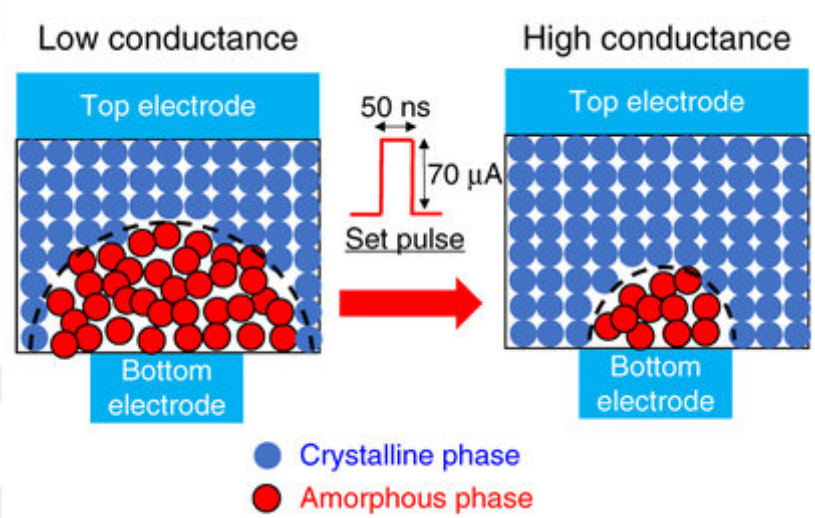

Figure 2.3: A PCM device illustration [16].

a stable resistance state that can be used for data storage [5].

2.1.1.3 Resistive Random Access Memory (ReRAM)

Resistive random access memory (ReRAM) memristors use materials such as hafnium oxide to create resistance changes in the device [18, 19]. These non-volatile memristors offer tunable conductance states, which can be incrementally increased or decreased. They are advantageous for in-memory computing applications, leveraging their ability to store and process information simultaneously. ReRAMs are also known for their scalability and lower power consumption compared to traditional memory types. [20] discusses a memory cell design with two serially connected memristors, addressing the sneak path problem and improving robustness against data errors caused by memristor variations.

2.1.1.4 Electrochemical Random Access Memory (EcRAM)

Electrochemical random access memory (EcRAM) devices utilize electrochemical reactions, typically involving the intercalation of ions such as lithium into

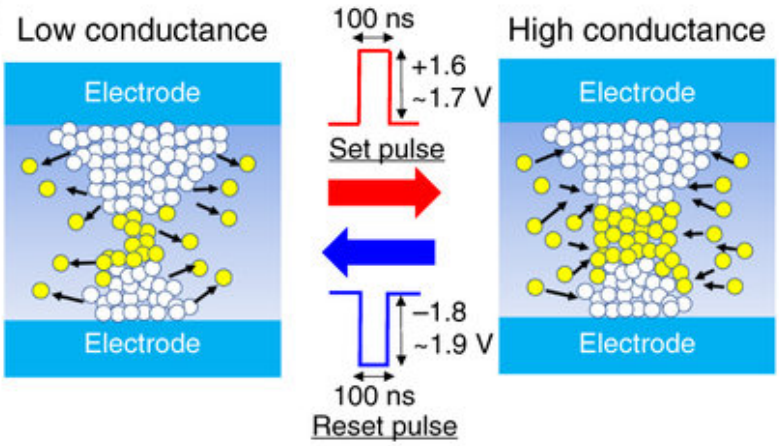

Figure 2.4: A ReRAM device illustration [16].

materials such as tungsten oxide. These memristors offer the advantage of near-symmetric switching, a large number of discrete states, and reduced stochasticity. Their three-terminal design allows for decoupled read and write operations, enhancing endurance and energy efficiency. EcRAMs are promising for analog AI applications due to their non-volatility and precise control over conductance changes. Ref. [21] study highlights electrochemical synaptic transistors for high-speed, symmetric analog programming with small variability, demonstrating compatibility with CMOS technology and scalability.

2.1.1.5 Capacitor-Cell Based Memristors

These memristors use capacitors, often in a crossbar configuration, integrated into CMOS technology [22]. They are known for their excellent linearity and symmetry in conductance changes. During operation, the capacitor is charged or discharged to modify the synaptic weight, emulating the behavior of biological synapses. These devices are notable for their potential to be highly linear and symmetric analog neural networks, making them strong candidates for advanced neuromorphic computing applications. [23] reports on memcapacitive devices with high dynamic ratios and energy-efficient operations, demonstrating the potential for high-performance scalable neuromorphic computing applications.

2.2 Hardware Implementations

In the literature, many hardware implementations of in-memory computing have been proposed to solve mathematical problems ranging from pattern recognition, regression, classification, and matrix equations. There are also some implementations for smaller neural network architectures. Most use a hybrid architecture where digital peripheral circuitry processes and transfers the signals between layers. We provide several examples of these studies in the following subsections.

2.2.1 MNIST Digit Classification

Ref. [24] proposes a five-layer convolutional neural network (CNN) implemented with memristor-based processing elements to classify MNIST digits. Each processing element is integrated with 2,048 one-transistor one-memristor (1T1R) cells arranged in crossbar arrays. This configuration ensures uniform and reliable analog switching behaviors essential for neural network operations. The hybrid training scheme used achieves an impressive recognition accuracy of 96.19% on the entire MNIST test dataset. Furthermore, to enhance computational efficiency, the convolutional kernels were replicated across three parallel memristor convolvers, effectively reducing the network latency by a factor of three. This implementation demonstrates the potential of memristor-based hardware systems to significantly improve the energy efficiency and processing speed of CNNs, making them a viable alternative to traditional digital computing architectures.

2.2.2 One-step Regression Fitting

Ref. [25] introduces a crossbar architecture resistive memory circuit capable of performing regression and classification tasks in one step. The authors illustrate an application of this technology using the Boston housing dataset for regression analysis. The circuit operates by computing the pseudoinverse of the input data

matrix within the memory, thereby enabling rapid and efficient training of traditional machine learning models such as linear and logistic regression. The authors report that this method significantly reduces the computational time and energy requirements compared to traditional iterative algorithms. The proposed circuit also shows promise in predicting housing prices with high accuracy and can be extended to other data-intensive applications.

2.2.3 Pattern Classification

The authors in [26] propose a scalable neuron circuit using conductive bridge random access memory for pattern recognition applications. Their work focuses on a circuit designed to recognize 3×5 pixel number patterns, which is fundamental for various image processing tasks. The conductive bridge random access memory (CBRAM)-based circuit takes advantage of the high density storage and low power characteristics of memristive devices. The authors demonstrate that the circuit can successfully identify number patterns with high accuracy and efficiency. This development underscores the potential of CBRAM technology in creating compact and energy-efficient pattern recognition systems, suitable for integration into larger neuromorphic computing frameworks.

2.2.4 Solving Matrix Linear Equation

Ref. [27] explores capabilities of resistive memory arrays in solving mathematical problems such as finding the solutions of 3×3 matrix linear equations or computing eigenvectors. Using a feedback circuit configuration, resistive memory arrays can perform these computations in a single step. This is achieved by mapping the matrix coefficients to the conductance states of the memristors and applying appropriate input currents. The system capitalizes on the inherent parallelism and analog storage capabilities of resistive memories, allowing efficient and rapid computations. The proposed method demonstrates significant improvements in

computational complexity and energy efficiency compared to conventional digital approaches, highlighting the potential of resistive memory arrays in advanced computational tasks.

2.2.5 Fully Hardware-Based Convolutional Neural Networks

Ref. [24] discusses developing and implementing a memristor-based CNN for image recognition problem, specifically using the MNIST dataset. The researchers developed high-yield, high-performance memristor crossbar arrays and introduced a hybrid training method to address device imperfections. The five-layer memristor-based CNN achieves an image recognition accuracy of over 95% while showing 110 times better energy efficiency than a Tesla V100 GPU.

2.3 Memristor Models

A general memristor can be modeled by showing the voltage-current relationship in the form given below

$$v(t) = R\left(w(t)\right) i(t) \tag{2.1}$$

$$\frac{dw}{dt} = f\left(w(t), i(t)\right) \tag{2.2}$$

where $v(t)$, $i(t)$, and $w(t)$ are the voltage, current, and internal state of the memristor, respectively. $w(t)$ is a state variable and R is a generalized resistance that depends on the device's internal state.

Multiple memristor models are presented in the literature, and we describe some of them in the following subsections.

2.3.1 Linear Ion Drift Model

This is one of the most popular models based on the linear movement of ions within a memristor film. It is known for its simplicity and intuitive understanding and is suggested in [4] by Hewlett-Packard (HP) labs. In this model, one assumption is that a device of physical width D contains two regions: one region of width w (which acts as a state variable) has a high concentration of dopants, and a second region of width $D-w$ is an oxide region. The region with the dopants has a higher conductance as compared to the oxide region; hence, the device is modeled as two resistors connected in series. The voltage-current relationship as a function of internal state and physical parameters is given by

$$v(t) = \left(R_{\text{on}} \frac{w(t)}{D} + R_{\text{off}} \left(1 - \frac{w(t)}{D} \right) \right) i(t), \tag{2.3}$$

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{\text{on}}}{D} i(t), \tag{2.4}$$

$$w(t) = \mu_v \frac{R_{\text{on}}}{D} q(t), \tag{2.5}$$

where R_{on} is the resistance when the width $w(t) = D$, and R_{off} is the resistance when $w(t) = 0$. The state variable $w(t)$ is constrained to the physical dimensions of the device, i.e., $[0, D]$. Table 2.1 shows the values of the parameters of this model.

Table 2.1: HP memristor model [4].

Parameter	**Value**	**Remark**
$R_{\text{OFF}}/R_{\text{ON}}$	$50 - 1250$	Ratio of HRS and LRS
μ_ν	$10^{-10} cm^2 s^{-1} V^{-1}$	Dopant mobility
D	$10nm$	Semiconductor film thickness

2.3.2 Non-Linear Ion Drift Model

The motivation for using the nonlinear ion drift model is to address the limitations of the linear ion drift model, which assumes a constant ion drift rate and fails to capture the nonlinearities and boundary effects observed in real memristors. This

often leads to issues such as border locking, where the state variable becomes stuck at the boundaries, and terminal state problems, where the model inaccurately represents the dynamics near extreme values. To overcome these challenges, the nonlinear ion drift model incorporates the Blackman window function [28], which offers a more detailed definition of boundary effects and results in more accurate simulations. The voltage-current relationship of the non-linear ion drift model is defined as

$$v(t) = M(x)i(t) = \left[R_{\text{on}}x + R_{\text{off}}(1-x)\right] i(t), \tag{2.6}$$

$$\frac{dx}{dt} = ki(t)f(x), \tag{2.7}$$

$$f(x) = j\left[a_0 - a_1\cos(2\pi x) + a_2\cos(4\pi x)\right], \tag{2.8}$$

where, $v(t)$ represents the voltage, $i(t)$ is the current, and $M(x)$ is the memristance, which depends on the state variable x. The resistances R_{on} and R_{off} correspond to the on and off states, respectively, while k is a constant derived from the physical properties of the memristor. The parameters a_0, a_1, a_2, and j are scaling factors used in the model. The parameters for this model are summarized in Table 2.2.

Table 2.2: Parameters of the memristor model [28].

Parameter	**Value**
R_{on}	0.1 kΩ
R_{off}	16 kΩ
k	$\frac{\mu_v R_{\text{on}}}{D^2}$
a_0	0.42
a_1	0.5
a_2	0.08
j	2

This nonlinear ion drift model with the Blackman window function provides a more accurate representation of the memristor's behavior by addressing the limitations of the linear ion drift model. However, the complexity of the Blackman window function may introduce additional computational overhead compared to simpler polynomial functions, potentially affecting the scalability and performance of the model.

2.3.3 Simmons Tunnel Barrier Model

The authors in [29] present the Simmons tunnel barrier model, which addresses asymmetric and non-linear switching behavior by incorporating an exponential relationship to the movement of ionized dopants. Unlike the linear drift model, which is modeled as two resistors in series, this model features a resistor in series with an electron tunnel barrier. Here, the state variable w represents the width of the tunnel barrier. The voltage-current relationship and the dynamic behavior of the switching process are described by the following equations

$$i = G(w, v)v, \tag{2.9}$$

$$\frac{dw}{dt} = f(w, i), \tag{2.10}$$

where, $G(w, v)$ is a generalized conductance that depends on voltage v and the state variable w.

The following equations describe the rate of change of the state variable w, as the function of state w and current i:

$$\frac{dw}{dt} = f_{\text{off}} \sinh\left(\frac{i}{i_{\text{off}}}\right) \exp\left(-\exp\left(\frac{w - a_{\text{off}}}{w_c} - \frac{i}{b}\right) - \frac{w}{w_c}\right) \quad \text{for } i < 0, \tag{2.11}$$

$$\frac{dw}{dt} = f_{\text{on}} \sinh\left(\frac{i}{i_{\text{on}}}\right) \exp\left(-\exp\left(-\frac{w - a_{\text{on}}}{w_c} - \frac{i}{b}\right) - \frac{w}{w_c}\right) \quad \text{for } i > 0, \tag{2.12}$$

where the parameters i_{on}, i_{off}, f_{on}, f_{off}, a_{on}, a_{off}, b, and w_c are determined through experimental fitting and characterize the detailed dynamics of the switching process. A summary of the parameters used in the model is provided in Table 2.3.

The Simmons tunnel barrier model simulates the nonlinear and asymmetric switching of memristive devices, providing insights into their behavior. However, the model's complexity requires extensive calibration and can be computationally intensive. Its sensitivity to parameter changes may affect predictive precision, and its reliance on experimental data limits its applicability to various devices.

Table 2.3: Parameters of the Simmons tunnel barrier model [29].

Parameter	Value
f_{off}	$3.5 \pm 1\,\mu m/s$
i_{off}	$115 \pm 4\,\mu A$
a_{off}	$1.20 \pm 0.02\,nm$
f_{on}	$40 \pm 10\,\mu m/s$
i_{on}	$8.9 \pm 0.3\,\mu A$
a_{on}	$1.80 \pm 0.01\,nm$
b	$500 \pm 70\,\mu A$
w_c	$107 \pm 4\,pm$

2.3.4 Threshold Adaptive Memristor Model (TEAM)

A simplified version of the Simmons tunnel barrier model, addressing some of its complexities while retaining essential features [30]. This model is adaptable to various practical memristive devices, balancing accuracy and computational efficiency. The model describes the dynamic behavior of the memristive device using the following equations:

$$v(t) = \left[R_{\text{on}} + \frac{R_{\text{off}} - R_{\text{on}}}{x_{\text{off}} - x_{\text{on}}}\left(x - x_{\text{on}}\right)\right] \cdot i(t), \tag{2.13}$$

$$\frac{dx}{dt} = \begin{cases} k_{\text{off}} \cdot \sinh\left(\frac{i}{i_{\text{off}}}\right) \cdot \exp\left(-\exp\left(\frac{x-a_{\text{off}}}{x_c} - \frac{i}{b}\right) - \frac{x}{x_c}\right) & \text{for } i < 0 \\ k_{\text{on}} \cdot \sinh\left(\frac{i}{i_{\text{on}}}\right) \cdot \exp\left(-\exp\left(-\frac{x-a_{\text{on}}}{x_c} - \frac{i}{b}\right) - \frac{x}{x_c}\right) & \text{for } i > 0 \end{cases} \tag{2.14}$$

where, $\frac{dx}{dt}$ is the rate of change of the state variable, i is the current through the memristor, x is the state variable representing the internal state of the memristor and k_{off}, k_{on}, i_{off}, i_{on}, a_{off}, a_{on}, b, and x_c are parameters determined experimentally.

The TEAM model provides accurate and computationally efficient modeling of memristive devices, capturing nonlinear and asymmetric switching behaviors. It is highly adaptable, fitting a wide range of practical memristive devices. However, the complexity of the model requires extensive experimental fitting and can be computationally demanding. The sensitivity to parameter variations can impact predictive accuracy, and the model's heavy reliance on experimental data for parameter determination limits its generalizability across different devices.

2.3.5 Voltage-Controlled Threshold Adaptive Memristor Model (VTEAM)

The VTEAM model is based on the derivative of an internal state variable. It combines the advantages of the TEAM model while utilizing a threshold voltage instead of a threshold current. The voltage-current relationship and the dynamic behavior of the state variable are given by the following equations:

$$i(t) = \left[R_{\text{on}} + \frac{R_{\text{off}} - R_{\text{on}}}{w_{\text{off}} - w_{\text{on}}} (w - w_{\text{on}}) \right]^{-1} v(t), \tag{2.15}$$

$$\frac{dw(t)}{dt} = \begin{cases} k_{\text{off}} \left(\frac{v(t)}{v_{\text{off}}} - 1 \right)^{\alpha_{\text{off}}} f_{\text{off}}(w), & 0 < v_{\text{off}} < v(t) \\ 0, & v_{\text{on}} < v(t) < v_{\text{off}} \\ k_{\text{on}} \left(\frac{v(t)}{v_{\text{on}}} - 1 \right)^{\alpha_{\text{on}}} f_{\text{on}}(w), & v(t) < v_{\text{on}} < 0 \end{cases}, \tag{2.16}$$

where the parameters f_{off}, f_{on}, k_{off}, k_{on}, v_{off}, v_{on}, α_{off} and α_{on} are determined by experimental fitting and describe the detailed dynamics of the switching process. The details are available in [31].

The VTEAM model has comparable advantages and drawbacks as the TEAM model, with the main difference being that it is suitable for voltage-controlled memristors.

2.4 Stochastic Inference Model for PCM Memristors

The stochastic model for PCM-type memristors, as introduced in [32, 33, 34, 35], is grounded in extensive data measurements collected from millions of devices. This model is developed for simulating memristive neural networks during inference tasks and accounts for the non-idealities inherent in analog in-memory computations. Rather than focusing on the detailed physics of the devices, the model abstracts the changes in conductance as normally distributed random variables, with their mean and variance empirically fitted to the observed data. This

abstraction captures the essential behavior of the devices while allowing for practical simulation of large-scale neural networks.

The relevance of this model to our work is significant, as it strikes a balance between computational efficiency and the realistic simulation of in-memory computing effects. By focusing on the stochastic nature of conductance changes rather than a full analog device simulation, the model provides a pragmatic approach that suits the demands of our investigations. With this motivation, in the following, we will detail the various noise sources modeled within this framework and explain how they influence the conductance values, contributing to the overall performance of memristive neural networks.

2.4.1 Programming Noise

This noise originates from discrepancies between the target and the actual conductance values programmed into the hardware. This is represented as an additive normal distribution with zero mean and standard deviation σ_{prog}. The conductance after programming g_{prog} is then given as

$$g_{\text{prog}} = g_T + \mathcal{N}\left(0, \sigma_{\text{prog}}\left(g_T\right)\right), \tag{2.17}$$

where $\sigma_{\text{prog}}\left(g_T\right)$ is defined as a quadratic function of target conductance g_T,

$$\sigma_{\text{prog}}\left(g_T\right) = \max\left(-1.1731 g_T^2 + 1.9650 g_T + 0.2635, 0\right). \tag{2.18}$$

The programmed conductance value, g_{prog}, subsequently deviates from the target value due to this programming noise. Figure 2.5 illustrates the correlation between the standard deviation of the programming noise and the normalized target conductance.

2.4.2 Drift Noise

Conductance values of PCM devices decay over time [35]. This drift is an intrinsic property of the phase-change material. Knowing the conductance g_{prog}

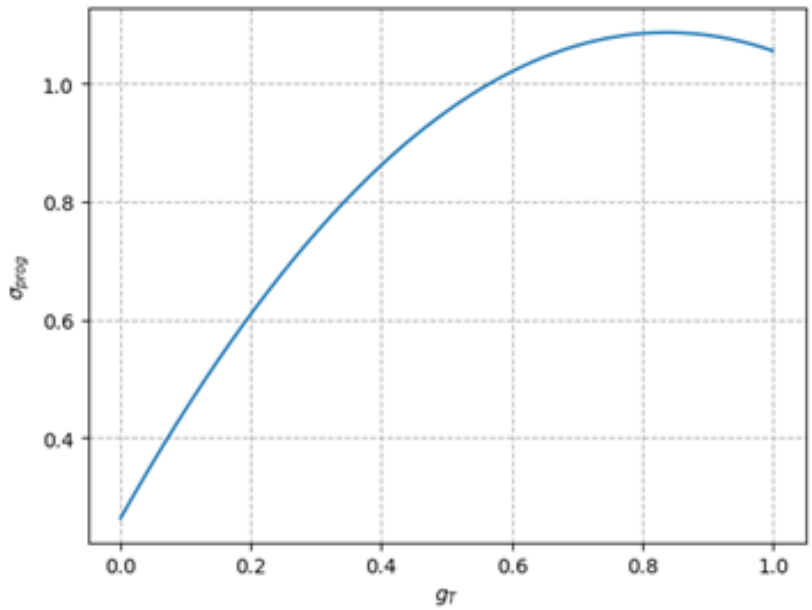

Figure 2.5: Standard deviation of programming noise against normalized target conductance.

programmed at time t_c, the conductance evolution $g_{\text{drift}}(t)$ can be modeled as a decaying exponential with exponent ν.

$$g_{\text{drift}}(t) = g_{\text{prog}} \left(\frac{t}{t_c} \right)^{-\nu}, \tag{2.19}$$

where ν follows a normal distribution whose mean μ_ν and σ_ν are the functions of target conductance g_T, and are defined as

$$\nu = \mathcal{N}\left(\mu_\nu\left(g_T\right), \sigma_\nu\left(g_T\right)\right), \tag{2.20}$$

$$\mu_\nu(g_T) = \min\left(\max\left(-0.0155 \log g_T + 0.0244, 0.049\right), 0.1\right), \tag{2.21}$$

$$\sigma_\nu(g_T) = \min\left(\max\left(-0.0125 \log g_T - 0.0059, 0.008\right), 0.045\right). \tag{2.22}$$

Figure 2.6 shows the relationship between the mean and standard deviation of the drift exponent against the normalized target conductance.

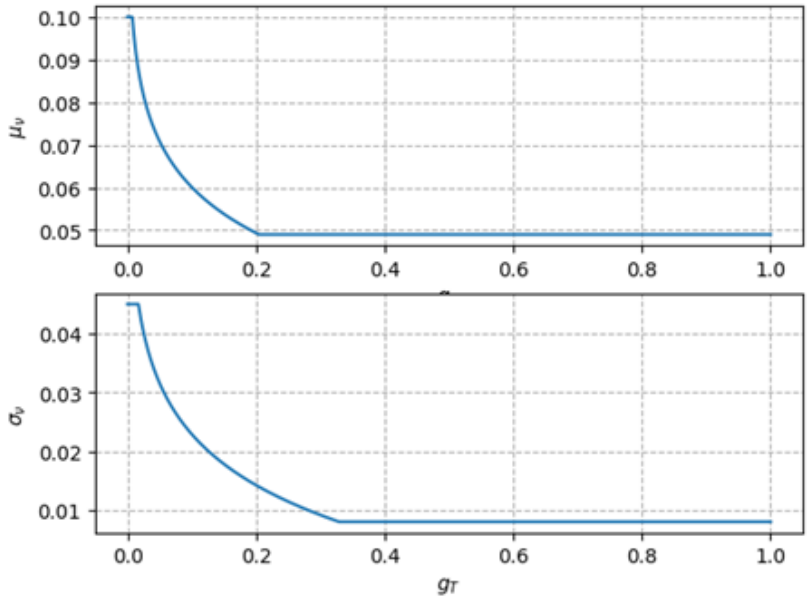

Figure 2.6: Mean and standard deviation of drift exponent against normalized target conductance.

2.4.3 Read Noise

This noise source appears during the execution of MVM operation with the in-memory computing hardware after weight programming. It stands for instantaneous fluctuations in hardware conductance due to the intrinsic noise characteristics of PCM devices, namely $1/f$ noise and random telegraph noise [33]. This is an additive Gaussian noise whose standard deviation is a function of the drift conductance defined in (2.19), a coefficient Q_s, and the current read time t. This noise then leads to further fluctuation in the total conductance value. The standard deviation $\sigma_{nG}(t)$ of the read noise and the final conduncatce value $g(t)$ are given as,

$$\sigma_{nG}(t) = g_{\text{drift}}(t)\, Q_s \sqrt{\log \frac{t + t_c}{2t_{\text{read}}}}, \tag{2.23}$$

$$g(t) = g_{\text{drift}}(t) + \mathcal{N}(0, \sigma_{nG}(t)). \tag{2.24}$$

2.4.4 Temperature Noise

Since phase-change materials are typically low-bandgap semiconductors, the thermally activated nature of electrical transport implies that the device's conductance becomes sensitive to the ambient temperature: an increase in ambient temperature results in a decrease in conductance and vice versa. Mathematically, temperature sensitivity follows the relation

$$g = g_{\text{prog}} \times \exp^{\frac{-E_a}{kT}}, \tag{2.25}$$

where E_a is the activation energy for thermal transport, k is the Boltzmann constant, and T is temperature in Kelvin.

2.4.5 Bipolar Asymmetry

When phase-change materials contact metallic electrodes, interfacial barriers can form, the size of which is dictated by both phase-change materials and electrode properties. If the barriers at the interfaces with the bottom and top electrodes differ, the current flow would become polarity dependent. Mathematically, bipolar asymmetry follows the relation

$$\alpha_{ij} = \frac{I_{ij(\text{pos})}}{I_{ij(\text{neg})}}, \tag{2.26}$$

where the current under a positive polarity read signal differs from the negative polarity read signal by factor α.

2.4.6 Conductance Mapping and Effect of Noise on Network Weights

To translate the weight values of the neural network into conductance terms, a differential configuration employing a pair of memristors is used. Each weight, represented as w_{ij}, is programmed onto a pair of PCM devices. Depending on

the sign of the weight, one device is programmed while the other is set to its minimum conductance state $g_{\min}$. The target conductance for the positive and negative memristors $g_{T_{\text{pos}}}$ and $g_{T_{\text{neg}}}$, respectively, is then determined by the following conditionals:

$$g_{T_{\text{pos}}} = \begin{cases} w\frac{g_{\max}-g_{\min}}{\max(|\mathbf{W}|)} & \text{if } w > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{2.27}$$

$$g_{T_{\text{neg}}} = \begin{cases} -w\frac{g_{\max}-g_{\min}}{\max(|\mathbf{W}|)} & \text{if } w < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.28}$$

After establishing target conductances, following noise models are applied,

$$g'_{T_{pos}} = \left(g_{T_{pos}} + \mathcal{N}\left(0, \sigma_{\text{prog}}\left(g_{T_{pos}}\right)\right)\right)\left(\frac{t}{t_c}\right)^{-\nu} + \mathcal{N}\left(0, \sigma_{nG}\left(t\right)\right), \tag{2.29}$$

$$g'_{T_{neg}} = \left(g_{T_{neg}} + \mathcal{N}\left(0, \sigma_{\text{prog}}\left(g_{T_{neg}}\right)\right)\right)\left(\frac{t}{t_c}\right)^{-\nu} + \mathcal{N}\left(0, \sigma_{nG}\left(t\right)\right). \tag{2.30}$$

These **noisy** conductance values are then reverse-mapped to obtain the noisy weights as,

$$w' = \frac{\left(g_{\max} - g_{\min}\right)\left(g'_{T_{pos}} - g'_{T_{neg}}\right)}{\max\left(|\mathbf{W}|\right)}. \tag{2.31}$$

Here, w' symbolizes the noisy weights after factoring in the various noise sources of the PCM memristive devices.

To observe the impact of the noise model, we use example weight matrices of size 1000×1000 with uniformly and normally distributed weights, and apply programming, drift, and read noise as explained in the PCM model in Equation (2.31).

Figure 2.7 demonstrates the impact of programming noise on the weight distribution, where the noise causes weight-dependent perturbations. The peak at 0 results from setting all negative conductance values to zero—if a conductance becomes negative after adding noise, it is forced to zero.

Figure 2.8 displays the effect of conductance drift over a one-year period following programming. This results in a contracted distribution due to the decaying nature of drift noise.

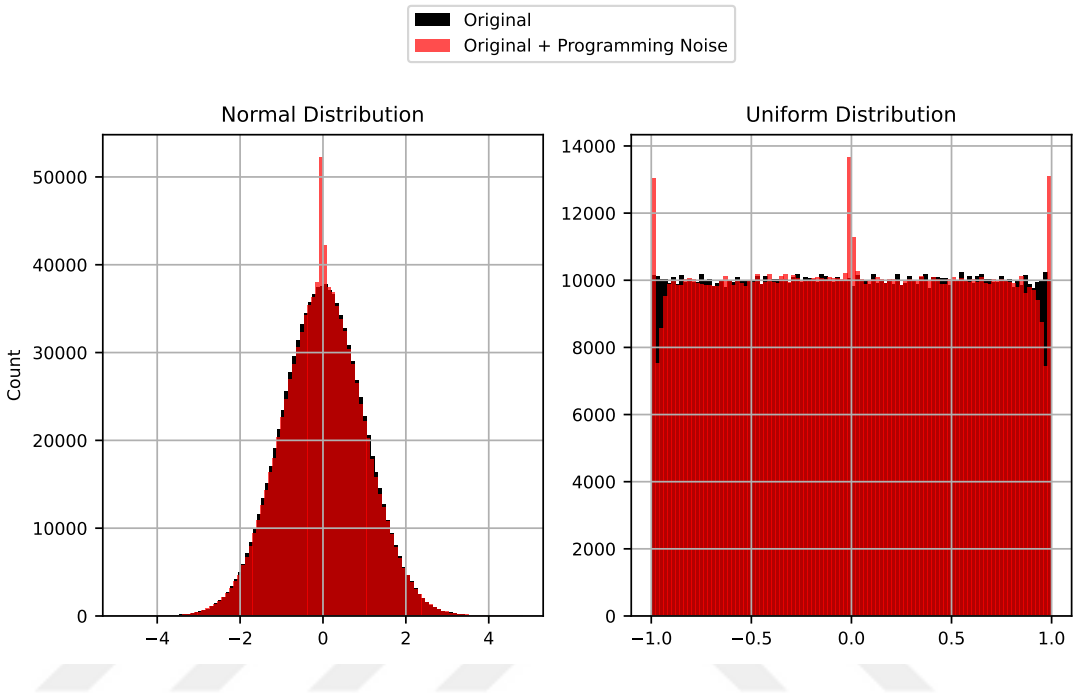

Figure 2.7: PCM programming noise effect.

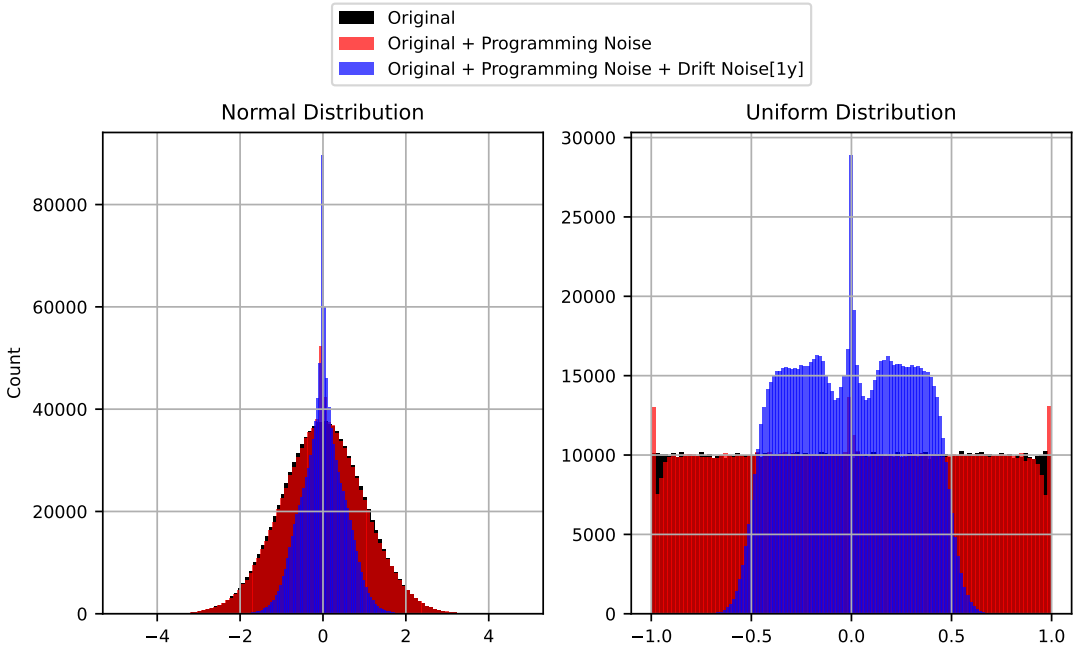

Figure 2.8: PCM programming + drift noise effect for 1 year.

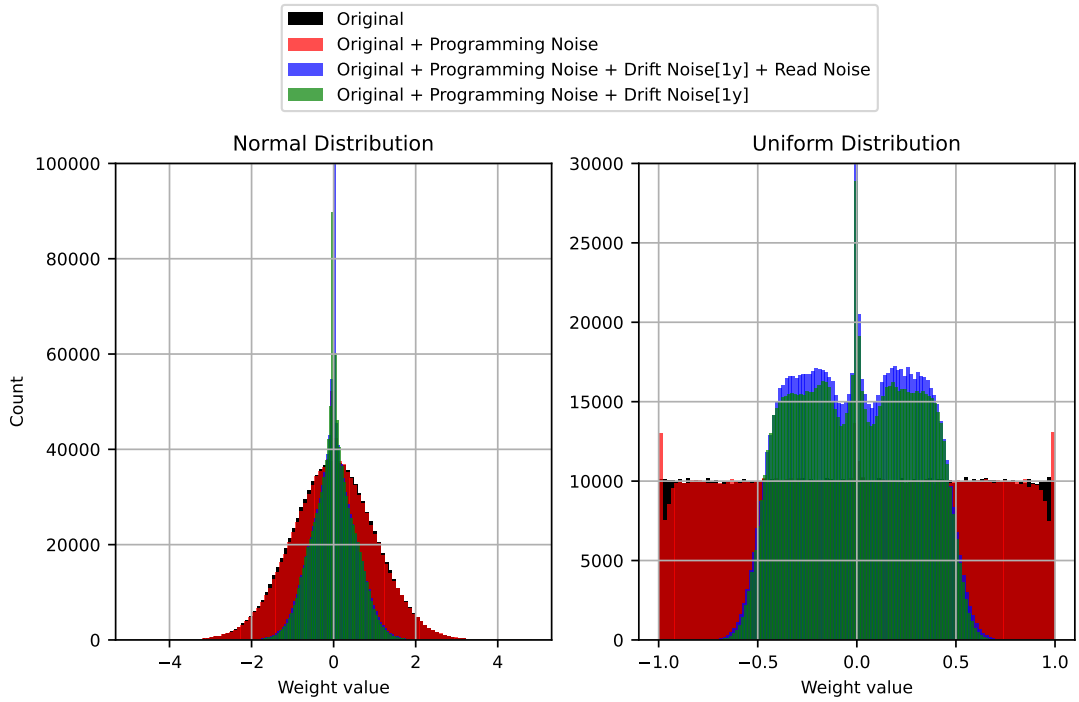

Figure 2.9: PCM programming + drift + read noise effect.

Figure 2.9 shows the commutative effect of programming, drift and read noise. Read noise is modeled as zero-mean additive Gaussian noise and is negligible compared to programming and drift noise.

It is important to note that we do not simulate the effects of temperature noise and bipolar asymmetry, as these are considered outside the scope of this study.

2.5 Simulation of Memristive Neural Networks

Simulating a memristive neural network requires simulating a crossbar architecture with input and output scaling and discretization (analog-to-digital converter (ADC)/digital-to-analog converter (DAC)), weight transfer, nonlinear effects, and the type of fault in a memristive device.

In general, we can classify existing simulators into two categories: architecture-level and circuit-level frameworks. Architecture-level simulation frameworks use behavioral models and analytical calculations to obtain the characteristics of different in-memory computing (IMC) architectures. Conversely, circuit-level simulators leverage circuit analysis techniques to analyze the functionality and performance of IMC-based neural NN circuits. Although architecture-level tools are faster than circuit-level frameworks, they typically omit essential design details, potentially leading to discrepancies between simulation and actual implementation results. On the other hand, full circuit simulations offer greater accuracy but at the expense of longer simulation times.

MNSim [36, 37] and NeuroSim [38] are architecture-level simulation frameworks that employ analytical models to estimate the power, latency, and area of IMC architectures. Validation results for MNSim indicate an error margin of approximately 5% in the calculation of power, energy, and latency compared to circuit simulations for a fully connected two-layer neural network. Both are suitable for only specific classes of memristive devices and multi layer perceptron (MLP) neural network architectures [39] and lack flexibility.

The IMAC-Sim [40] allows simulating for an extensive exploration of design parameters, offering automated evaluations of circuit accuracy, energy use, and latency based on user-defined settings. The simulator also incorporates parasitic resistance and capacitance calculations in IMC architectures, along with techniques to overcome related reliability issues. The simulator is based on HSpice [41].

CrossSim [42] is a Python-based, GPU-accelerated crossbar simulator developed to model analog in-memory computing for any application that relies on matrix operations: neural networks, signal processing, solving linear systems, and many more.

In the subsequent chapters, we explore applications of in-memory computing in wireless communications and focus on the benefits and effects of analog computations rather than developing a high-fidelity memristive neural network simulator. Consequently, instead of opting for an existing circuit-level simulator, we develop a custom simulator using PyTorch [43]. In our implementation, we modify the default PyTorch layers by perturbing the layer parameters according to the PCM inference model for forward pass as detailed in Section 2.4. Additionally, we implement methods to convert an existing network to its analog equivalent network by replacing existing layers with our custom layers. This approach gives us the flexibility to modify training and inference as needed and we can investigate the effect of analog computations on existing pre-trained networks. Based on real measurements, the PCM model effectively captures the non-ideal effects of memristive devices without requiring a full circuit-level simulation. For this reason, we use it for our investigations of in-memory computing in wireless communications in the subsequent chapters. Specifically, this approach provides us with the flexibility to modify the inference and training phase without introducing a huge implementation complexity.

2.6 Chapter Summary

This chapter reviewed foundational concepts and key literature on DNNs, in-memory computing, and memristor technology, emphasizing the need for energy-efficient hardware for DNN operations. It highlights the potential of in-memory computing to enhance energy efficiency and the advantages of memristors due to their low latency and energy efficiency. The chapter discusses various memristor technologies, noise models for PCM-type memristors, and simulation frameworks

for memristive neural networks, showcasing the potential of memristor-based systems to improve processing speed and energy efficiency in complex applications.

Chapter 3

Cellular Network Traffic Prediction using Memristive Neural Networks

Forcasts predict that number of wireless cellular connected devices will reach 150 billion by 2030 [44]. Predicting cellular network traffic at base stations is essential for managing and optimizing telecommunications infrastructure [45]. It is fundamental for capacity planning, allowing network operators to anticipate demand peaks and avoid over investment. This foresight facilitates load balancing, ensuring an equitable distribution of network resources and preventing congestion. Crucially, traffic predictions aid in efficient resource allocation, notably in bandwidth and power, enhancing network efficiency and reducing operational costs. In addition, maintaining a high quality-of-service (QoS) is directly related to the ability to predict traffic volumes, especially in areas with fluctuating demand. This becomes increasingly relevant with the integration of emerging technologies such as 5G and the IoT, where network demands are ever-evolving. Predictive insights also guide the scheduling of network maintenance and upgrades, minimizing service disruptions. Accurate traffic prediction at base stations is a tool for current network optimization and a necessity for futureproofing cellular networks in an increasingly connected world.

Building on this critical need for accurate traffic prediction, our study, motivated by SUPERCOM's GLOBAL AI FOR 5G CHALLENGE (2022)[46], focuses on the predictive analysis of network traffic using an unlabeled real dataset from unidentified long-term evolution (LTE) users collected from a base station (BS) in Barcelona [47]. We study an analog implementation of cellular network traffic prediction problem by employing the PCM memristor model to simulate the neural network inference using in-memory computing. We study the effect of intrinsic noise in the PCM devices on prediction performance.

In addition, we study and propose robust training techniques to improve the performance loss due to analog computations. We see an improvement in performance when the network is exposed to weight noise during the training which makes it robust to the PCM noise experienced during inference.

The rest of the chapter is organized as follows. Section 3.1 describes the original cellular network traffic prediction problem and our contribution in realizing an analog implementation. Section 3.2 describes the robust training method. In Section 3.3 we present our simulation setup, dataset description, network architecture and numerical results.

3.1 Cellular Network Traffic Prediction

As described earlier, cellular network traffic prediction is extremely important for both service and infrastructure providers to ensure a stable network service with a guaranteed QoS. For example, an accurate estimate of future network activity can help avoid network traffic jams by reallocating traffic demand from busy towers to free ones.

Recent research has focused on modeling cellular network traffic prediction as a time series forecasting problem [48, 49, 50, 51]. Specifically, Ref. [52] proposes an end-to-end traffic demand forecasting framework based on deep leaning (DL). Here, we study the cellular network traffic prediction problem based on the

framework and dataset of [46]. We describe the dataset, dataset pre-processing and the network architecture used below.

Dataset: The dataset for this investigation is sourced from the ElBorn site and comprises 5243 samples. These samples have been divided so that 80% of the total samples, equating to 4195 samples, have been utilized for the training set, while the remaining 20%, or 1048 samples, have been reserved for the validation and test sets.

The dataset comprises real LTE physical downlink control channel (PDCCH) measurements obtained from three locations in Barcelona, Spain, using IMDEA-OWL [53], a free and open-source LTE control channel decoder developed by IMDEA Networks Institute [54]. The dataset provides several features averaged every 120 seconds during the measurements.

Table 3.1 lists available features in the dataset. Out of the 11 available features, five, namely *rnti_count*, *rb_up*, *rb_down*, *mcs_up*, and *mcs_down* are chosen as input features while *rnti_count*, *rb_up*, *rb_down*, *up*, and *down* are chosen as output features we want to predict. All missing data samples are replaced with 0 values. The input and output data are split into training and validation sets with an $8:2$ ratio and scaled between $[-1:1]$ for efficient training convergence.

Table 3.1: Feature Descriptions

Feature	**Description**
down	Transport block size in the downlink
up	Transport block size in the uplink
mcs_down	Modulation and coding scheme (download)
mcs_up	Modulation and coding scheme (upload)
rb_down	Number of allocated resource blocks (download)
rb_up	Number of allocated resource blocks (upload)
rtni	Radio network temporary identifiers count
mcs_down_var	Variance of mcs_down
mcs_up_var	Variance of mcs_up
rb_down_var	Normalized variance of rb_down
rb_up_var	Normalized variance of rb_up

Preprocessing: The data samples are reshaped into time series objects. In

this transformed configuration, each data input sample captures the history of the preceding 10 time samples. The stride, or step size, has been set at 1, implying that successive data samples share a 90% overlap with their immediate predecessor. The subsequent time instance immediately following each 10-sample input window determines the target data for each input sample. This process of selecting the target data for the input samples is graphically illustrated in Figure 3.1. The robust structure of this data transformation, offering a high degree of overlap and a clear methodology for selecting target data, lends itself well to the accuracy and dependability of future data analysis.

Network Architecture: A CNN type network is used to predict the network traffic. It comprises four convolution layers followed by an average pooling layer and a dense layer as output. The complete specification of the network architecture is given in Table 3.2.

Table 3.2: Neural network architecture for cellular network traffic prediction.

Layer Type	**Filters**	**Kernel Size**	**Padding**	**Activation**
Convolutional	16	[16, 3]	same*	ReLU
Convolutional	16	[3, 5]	same	ReLU
Convolutional	32	[8, 3]	same	ReLU
Convolutional	32	[4, 3]	same	ReLU
Average Pooling		[2, 1]		
Dense (800 x 5)				tanh

* ensures same input and output dimensions.

3.1.1 Analog Cellular Network Traffic Prediction

DL based solutions for cellular network traffic prediction provide good performance; however, they are often computationally expensive and challenging to implement in real-time applications due to the complexity and size of the neural networks. To address the problem of real-time implementation for cellular network traffic prediction, we investigate moving the inference phase of DL to an equivalent analog in-memory computation using memristors. Memristors configured in a crossbar architecture can perform a MVM operation in a single time

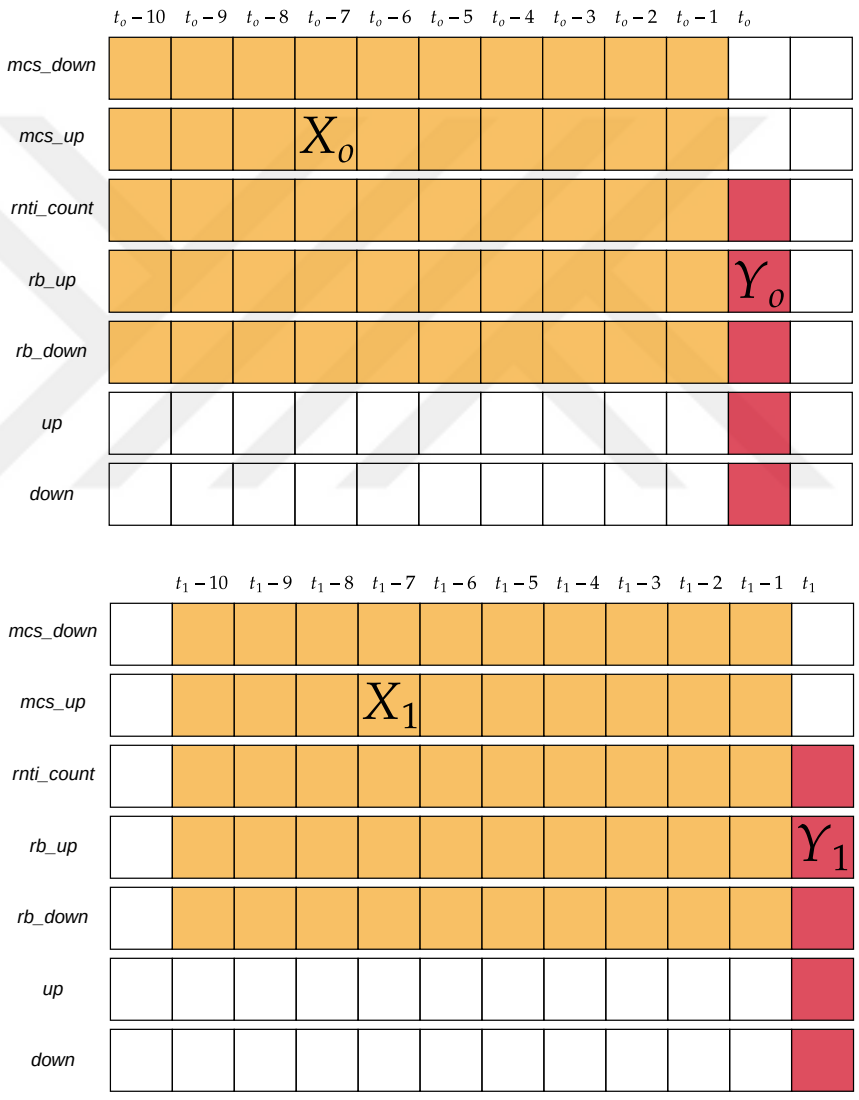

Figure 3.1: Data preprocessing.

step using KVL and hence are excellent neural network accelerators for real-time applications.

To simulate the DL inference using analog in-memory computing, we use the PCM model described in Section 2.4. Each layer of the network is converted to an equivalent analog layer. The conversion process involves two main steps: (1) programming of the parameters and (2) drifting of the parameters. In the programming phase, the parameters of the trained network are subjected to programming noise. This results in an equivalent analog model that simulates the effects of the difference between the programmed and desired conductance values. This effect is modeled as a Gaussian random variable with conductance-dependent mean and variance. Conductance drift comes into effect as time passes after the programming phase. The memristive devices undergo a drift in the programmed conductance values and decay over time. This effect is modeled as a multiplicative drift exponent with coefficients determined empirically. A third noise source is read noise, which is applied each time the network is used for inference. This is also modeled as a zero mean additive Gaussian noise.

To apply the PCM noise to the network parameters, each parameter, represented as w_{ij}, is programmed onto a pair of PCM devices. Depending on the sign of the parameter, one device is programmed while the other is set to its minimum conductance state $g_{\min}$. The target conductance for the positive and negative memristors $g_{T_{\text{pos}}}$ and $g_{T_{\text{neg}}}$, respectively, is then determined by the following conditionals:

$$g_{T_{\text{pos}}} = \begin{cases} w \frac{g_{\max} - g_{\min}}{\max(|\mathbf{w}|)} & \text{if } w > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

$$g_{T_{\text{neg}}} = \begin{cases} -w \frac{g_{\max} - g_{\min}}{\max(|\mathbf{w}|)} & \text{if } w < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

After establishing target conductances, following noise models are applied

$$g'_{T_{pos}} = \left(g_{T_{pos}} + \mathcal{N}\left(0, \sigma_{\text{prog}}\left(g_{T_{pos}}\right)\right)\right)\left(\frac{t}{t_c}\right)^{-\nu} + \mathcal{N}\left(0, \sigma_{nG}\left(t\right)\right), \tag{3.3}$$

$$g'_{T_{neg}} = \left(g_{T_{neg}} + \mathcal{N}\left(0, \sigma_{\text{prog}}\left(g_{T_{neg}}\right)\right)\right)\left(\frac{t}{t_c}\right)^{-\nu} + \mathcal{N}\left(0, \sigma_{nG}\left(t\right)\right). \tag{3.4}$$

These noisy conductance values are then reverse-mapped to obtain the noisy parameters as,

$$\mathbf{w}' = \frac{\left(g_{\max} - g_{\min}\right)\left(g'_{T_{pos}} - g'_{T_{neg}}\right)}{\max\left(|\mathbf{w}|\right)}. \tag{3.5}$$

Here, $\mathbf{w}'$ symbolizes the noisy parameters after factoring in the various noise sources of the PCM memristive devices. Details of the PCM noise model are available in Section 2.4.

Figure 3.2 illustrates the mean squared error (MSE) and mean absolute error (MAE) loss of digital and analog implementation for the cellular network traffic prediction problem. Converting the network to analog adds the effect of programming and read noise, but it does not significantly affect prediction performance and exhibits similar loss values. However, when the memristive neural network experiences drift noise for various time intervals, both MSE and MAE loss increase, resulting in a decrease in prediction performance. The decaying nature of drift noise causes the parameter distribution to contract by forcing them towards zero and hence a degraded performance is observed.

3.2 Robust Training

As seen from Figure 3.2, when a trained neural network is converted to an equivalent analog neural network by applying the transformations using the PCM noise model, we observe a huge performance loss, especially with time drift. To address this issue, we introduce and experiment with a few robust training techniques and study their performance.

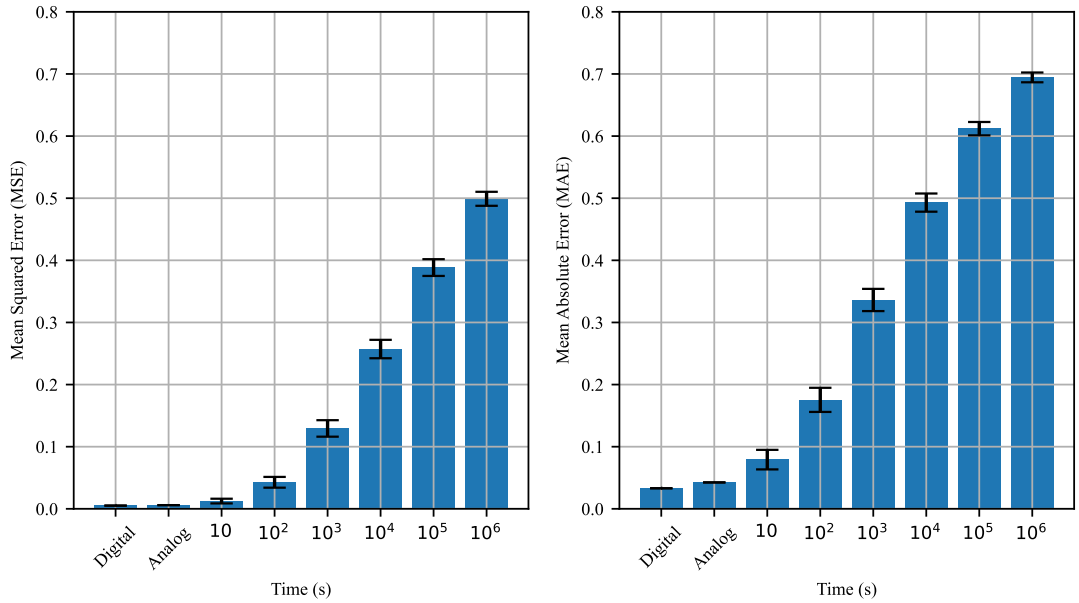

Figure 3.2: MSE and MAE for different time instances for digital and a analog inference.

In the following section, we describe the process of robust training by first understanding the standard neural network training for a simpler example neural network of Figure 3.3 and highlight the differences to convert it to robust training. Neural network training using backpropagation and gradient descent involves several steps:

Initialization: Weights and biases of the neural network are initialized randomly.

Forward Pass: During the forward pass, the input data is sequentially passed through each layer of the neural network to compute the output. Each layer transforms its input using a parameter matrix (weights) and applies an activation function to produce an output. These outputs serve as inputs to the subsequent layer, propagating the data through the network until the final output is generated.

For a fully connected neural network, the weights for each layer are represented by matrices. For example, in the given sample network, the weight matrices for layers $l-1$ and l are denoted as $\mathbf{w}^{(l-1)}$ and $\mathbf{w}^{(l)}$ respectively. These matrices

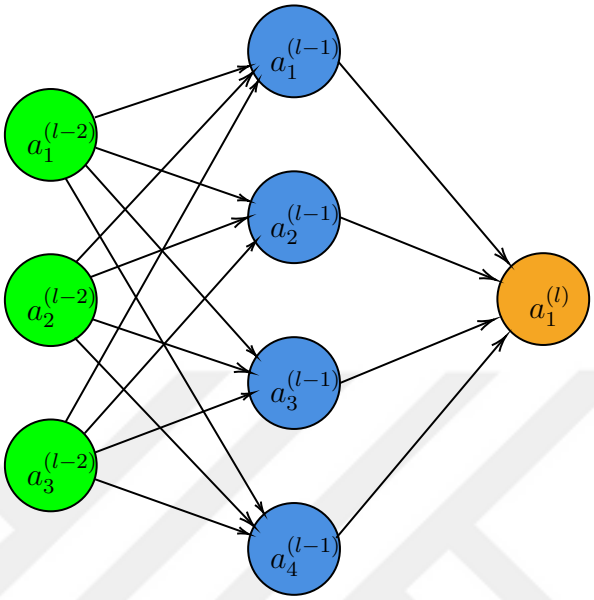

Figure 3.3: A sample fully connected neural network.

define the connections between the neurons of the previous layer and the current layer. The activations from the previous layers, denoted by $\mathbf{a}^{(l-2)}$ and $\mathbf{a}^{(l-1)}$, are multiplied by these weight matrices to produce the intermediate outputs $\mathbf{z}^{(l-1)}$ and $\mathbf{z}^{(l)}$.

$$\mathbf{w}^{(l-1)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}, \mathbf{w}^{(l)} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \\ w_{41} \end{bmatrix}, \tag{3.6}$$

$$\mathbf{a}^{(l-2)} = \begin{bmatrix} a_1^{(l-2)} & a_2^{(l-2)} & a_3^{(l-2)} \end{bmatrix}, \mathbf{a}^{(l-1)} = \begin{bmatrix} a_1^{(l-2)} & a_2^{(l-2)} & a_3^{(l-2)} & a_4^{(l-2)} \end{bmatrix}. \tag{3.7}$$

The following equations show that the intermediate outputs, $\mathbf{z}^{(l-1)}$ and $\mathbf{z}^{(l)}$, are calculated by performing matrix multiplication between the activations from the previous layers and the corresponding weight matrices. The resulting values are then passed through an activation function, such as rectified linear unit (ReLU),

to produce the activations $\mathbf{a}^{(l-1)}$ and $\mathbf{a}^{(l)}$ for the current layer:

$$\mathbf{z}^{(l-1)} = \mathbf{a}^{(l-2)}\mathbf{w}^{(l-1)}, \tag{3.8}$$

$$\mathbf{z}^{(l)} = \mathbf{a}^{(l-1)}\mathbf{w}^{(l)}, \tag{3.9}$$

$$\mathbf{a}^{(l-1)} = \mathrm{ReLU}\left(\mathbf{z}^{(l-1)}\right), \tag{3.10}$$

$$\mathbf{a}^{(l)} = \mathrm{ReLU}\left(\mathbf{z}^{(l)}\right). \tag{3.11}$$

Loss Calculation: After obtaining the predicted output, the next step is to compute the loss. As an example, we consider a Mean Squared Error (MSE) loss function given as:

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(a_i^{(l)} - y_i\right)^2, \tag{3.12}$$

where L is the calculated loss, n is the number of elements in the output vector, and $a_i^{(l)}$ and y_i are the predicted value and the true label for the ith dataset element, respectively.

Backpropagation: In backpropagation, the gradient of the loss function is calculated with respect to each network parameter using the chain rule. This process works by calculating the gradient for each layer, beginning from the output layer and working backward toward the input layer. The gradient indicates how much each parameter value contributes to the overall loss, and it is used to update the network parameters during training.

For the example network in Figure 3.3, the gradient of the loss L with respect to the parameter vector $\mathbf{w}^{(l)}$ is calculated using the chain rule as follows:

$$\frac{\partial L}{\partial \mathbf{w}^{(l)}} = \frac{\partial L}{\partial \mathbf{a}^{(l)}}\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}}\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{w}^{(l)}}. \tag{3.13}$$

This equation illustrates how the chain rule decomposes the gradient of the loss with respect to the weights into three components: the gradient of the loss with respect to the activation $\mathbf{a}^{(l)}$, the gradient of the activation with respect to the pre-activation output $\mathbf{z}^{(l)}$, and the gradient of the pre-activation output with respect to the weights $\mathbf{w}^{(l)}$.

If $\mathbf{a}^{(l)}$ is a ReLU activation, Equation (3.13) can be further decomposed as follows:

$$\frac{\partial L}{\partial \mathbf{a}^{(l)}} = \frac{2}{n}\left(\mathbf{a}^{(l)} - \mathbf{y}\right), \tag{3.14}$$

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \begin{cases} 1 & \text{if } \mathbf{z}^{(l)} > 0 \\ 0 & \text{if } \mathbf{z}^{(l)} < 0 \end{cases}, \tag{3.15}$$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{w}^{(l)}} = \mathbf{a}^{(l-1)}. \tag{3.16}$$

In this decomposition, $\frac{\partial L}{\partial \mathbf{a}^{(l)}}$ represents the gradient of the loss with respect to the output of the current layer, which can be directly calculated from the difference between the predicted output $\mathbf{a}^{(l)}$ and the true label $\mathbf{y}$. The term $\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}}$ corresponds to the derivative of the ReLU activation function, which is 1 when the input is positive and 0 otherwise. Lastly, $\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{w}^{(l)}}$ denotes the gradient of the linear transformation with respect to the weights, which is equivalent to the activations from the previous layer $\mathbf{a}^{(l-1)}$.

Together, these gradients are multiplied to yield the gradient of the loss with respect to the weights, which is then used to update the weights during the optimization process, typically using gradient descent or a similar algorithm.

Weight Update: After calculating the gradients, the network parameters $\mathbf{w}^{(l)}$ are updated to new values $\mathbf{w}^{(l)}_{\text{new}}$ by subtracting a scaled version of the gradient $\frac{\partial L}{\partial \mathbf{w}^{(l)}}$, as shown below:

$$\mathbf{w}^{(l)}_{\text{new}} = \mathbf{w}^{(l)} - \eta \frac{\partial L}{\partial \mathbf{w}^{(l)}}, \tag{3.17}$$

$$\mathbf{w}^{(l)}_{\text{new}} = \mathbf{w}^{(l)} - \eta \frac{2}{n}\left(\mathbf{a}^{(l)} - \mathbf{y}\right)\mathbf{a}^{(l-1)}, \tag{3.18}$$

where η is the learning rate, a hyperparameter that controls the step size of the update.

Repeat: The forward pass, loss calculation, backpropagation, and parameter update steps are iteratively repeated over multiple epochs until the loss function converges to a minimum or a predefined stopping criterion is satisfied.

Prediction: After the network has been trained, it is used to make predictions on new, unseen data. During this phase, the input data is passed through the network, where each layer applies its learned parameters(weights) and activation functions to produce the output. Depending on the task, the final output represents the predicted value or class label, such as classification or regression.

Robust Training: To enhance the robustness of network training, we modify the standard forward pass by introducing random noise to the network parameters. This means that during the forward pass, instead of using fixed parameters, we add a small random noise term to each parameter in every layer. This approach helps the network become resilient to variations in the parameter values, such as those caused by PCM noise.

The noisy parameters after noise addition are given by:

$$\mathbf{w}_{\text{noisy}}^{(l-1)} = \mathbf{w}^{(l-1)} + \mathcal{N}(0, \sigma), \tag{3.19}$$

where σ is the noise standard deviation and is tuned to optimize the performance.

These noisy weights are then used to compute the forward pass and the corresponding loss L_{noisy}, analogous to the standard loss calculation. The training process proceeds as usual, using this noisy loss to compute gradients and update the weights. This method makes the network more robust, as it learns to perform well even when subjected to small perturbations to the parameter values.

3.2.1 Different Types of Robust Training

The proposed robust training methods differ in the type of noise added to the forward pass computations. We experimented with three methods with distinct noise types: 1) PCM noise, 2) multiplicative log-normal noise, and 3) additive white Gaussian noise.

3.2.1.1 PCM Noise

As we use a stochastic model to simulate neural network inference on PCM devices, the first model we have used is PCM noise. We apply the model for zero drift time and the resulting weight transformation is given by

$$g(w) = w + \mathcal{N}(0, \sigma_{\text{prog}}) + \mathcal{N}(0, \sigma_{\text{read}}). \tag{3.20}$$

Here, σ_{prog} and σ_{read} are programming and read noise respectively. This model is described in detail in Section 2.4.

3.2.1.2 Multiplicative Log-Normal Noise

As the weights of the network decay in magnitude with time and our objective is to minimize the degradation with time, it may be useful to train the network with a similar multiplicative noise [55] as experienced during inference to improve the robustness of the model during inference. Here, the noise function $g(w)$ is given by

$$g(w) = w \cdot e^{\sigma Z}. \tag{3.21}$$

Here, σ is a hyperparameter that controls noise strength, while Z is a standard normal random variable.

3.2.1.3 Additive White Gaussian Noise (AWGN)

The authors in [56] use AWGN to improve the performance of memristive neural networks. We use a similar approach and add an additive Gaussian noise to network parameters during training. For this type of noise, the noise function is represented as

$$g(w) = w + \mathcal{N}(0, \sigma). \tag{3.22}$$

Here, σ is the standard deviation of the additive Gaussian noise being added.

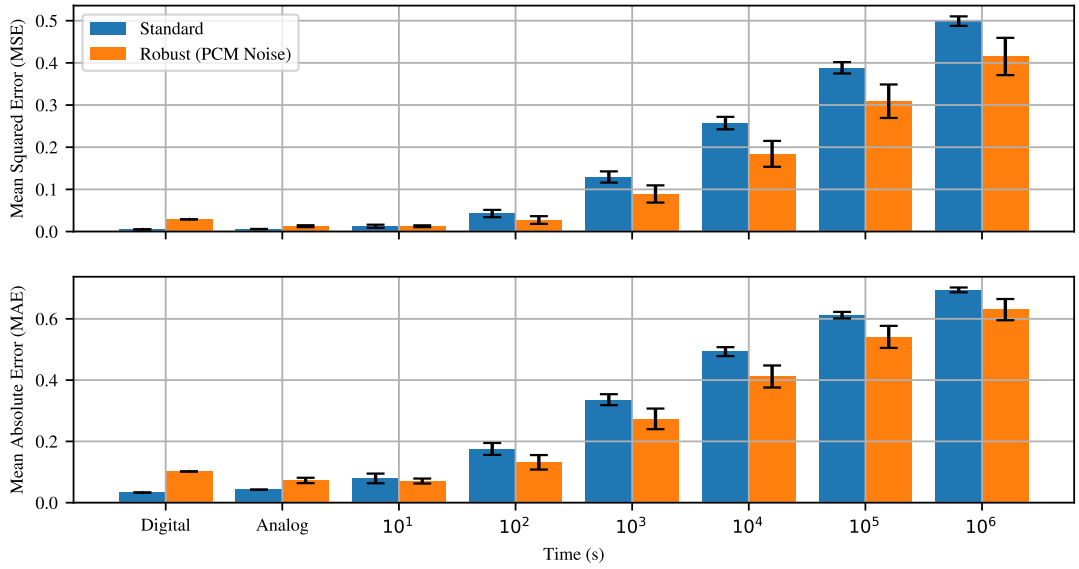

Figure 3.4: MSE and MAE for standard and robust training with PCM noise.

3.3 Numerical Results

In this section, we present the outcomes of our experimentation, focusing on the MSE and MAE on the test dataset after transforming the model into an analog network using the PCM noise model described in Section 2.4.

We conducted experiments with standard training and robust training using the three types of noises described in Section 3.2. We trained the network using the Adam optimizer with the MSE loss objective function. We save the best model by computing the average loss of 10 inferences and save the model with the lowest test loss during training. Averaging is done to get a better estimate of loss as the noise models are stochastic in nature. We trained the network for a maximum of 1000 epochs for each robust training case while stopping early if the test loss has not improved for 100 epochs.

Figure 3.4 shows the results of using the PCM noise model with zero drift during training. We do not see significantly improved model performance in terms of MSE and MAE. We believe that this could be because the PCM noise is weight dependent and affects differently depending on the magnitude of the

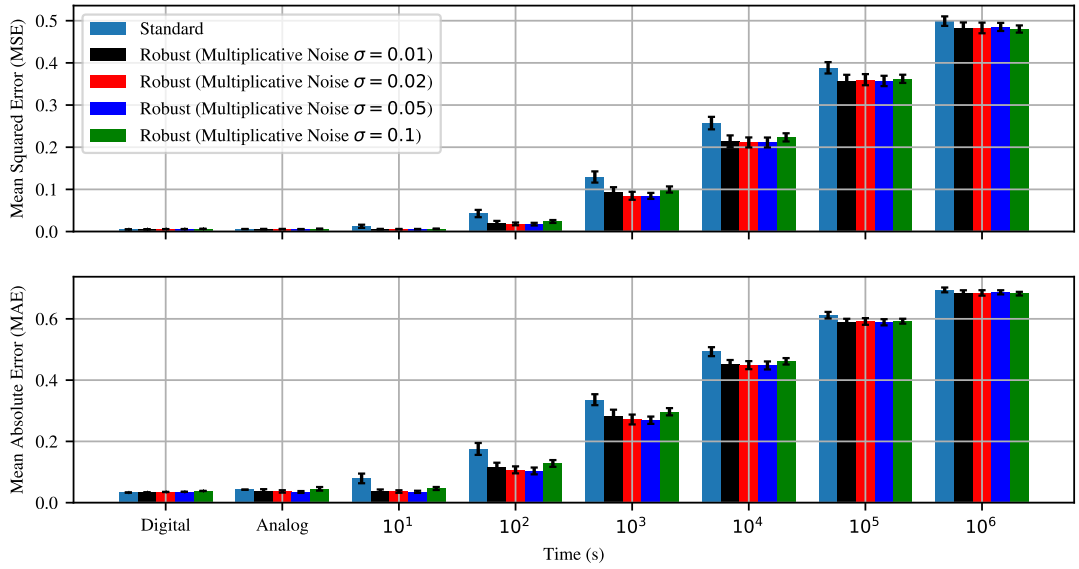

Figure 3.5: MSE and MAE for standard and robust training with multiplicative log-normal noise $\sigma = [0.01, 0.02, 0.05, 0.1]$.

weights.

As we optimize the performance for time drift, we add multiplicative log-normal noise with parameter $\sigma \in \{0.01, 0.02, 0.05, 0.1\}$. Being multiplicative in nature, this is similar to the conductance drift effect in PCM devices. We observe a slightly better performance agreeing with our insight, but the performance gain is still not very significant. Also, changing σ does not have significant effect on performance, with 0.05 looking marginally better than others. MSE and MAE with different values of σ against a range of drift times are shown in Figure 3.5.

The third method adds Gaussian noise to the weights during training. This approach increases the robustness of the system against conductance drift, improving the accuracy of the prediction. MSE and MAE are shown in Figure 3.6, which compares standard training against additive Gaussian noise based robust training. We observe an improvement in the prediction accuracy by a factor of up to 2 for later time instances, albeit at the cost of a negligible performance drop for earlier time instance values. This trade-off underlines the model's improved resilience against weight decay over extended periods, which is crucial for

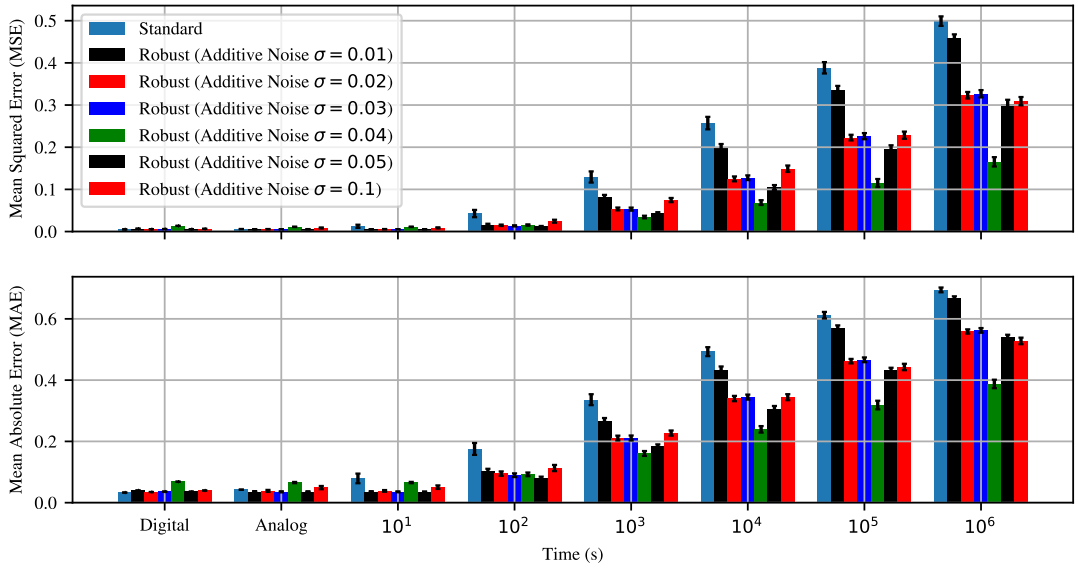

Figure 3.6: MSE and MAE for standard and robust training with PCM noise.

long-term, accurate network traffic prediction.

In Figure 3.7, we compare the best of all three robust training methods described previously. We observe that using additive Gaussian noise with $\sigma = 0.04$ gives the minimum MSE and MAE for the test dataset over the entire drift time range.

The results highlight the performance implications of transitioning from a noise-free digital model to a more realistic analog model that accounts for the noise characteristics of PCM devices.

To effectively predict cellular traffic using a neural network, we must recognize that loss and error values alone might not fully capture performance. It is important to analyze individual features to understand their significance and to identify where larger differences between predicted and actual values are acceptable.

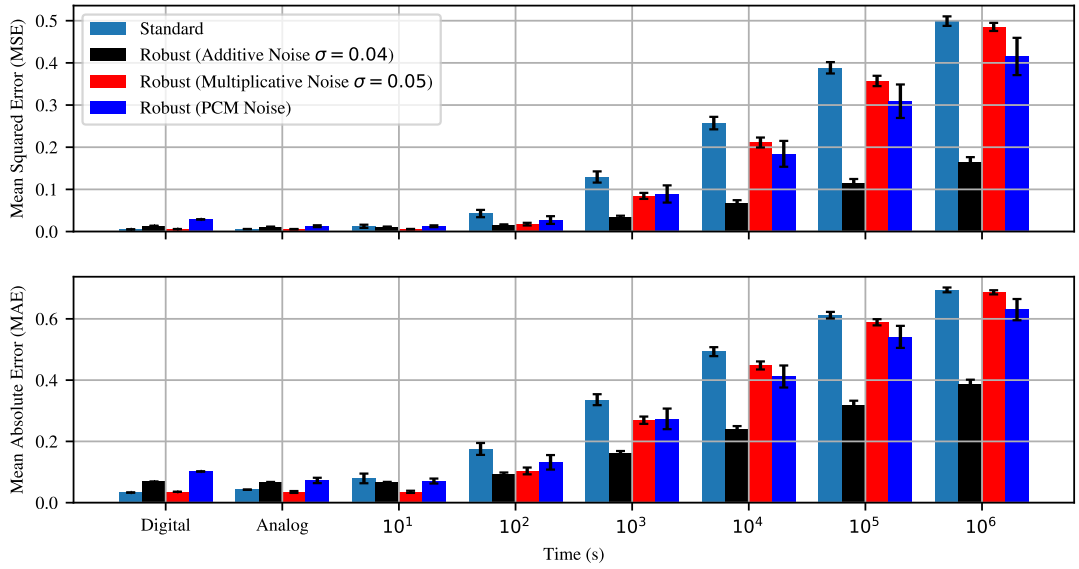

Figure 3.7: MSE and MAE for standard and robust training.

3.4 Energy Consumption

In Section 3.3, we evaluated the performance of 5G network traffic prediction using a CNN architecture memristive neural network. One of the benefits of using PCM devices for an analog implementation of the neural network is their fast inference and energy efficiency. The energy consumption depends on the technology used to fabricate the analog neural network inference chips. For our simplified model, we can estimate the energy consumption by modeling the energy consumed by the PCM devices.

In this section, we investigate the energy consumption and efficiency of deploying a neural network model on analog hardware, particularly focusing on the implications for battery-powered devices. This is crucial for understanding the viability of analog computation in portable electronics and IoT devices, where energy efficiency is paramount. We use memristors as analog devices, as they are naturally well suited for MVM operation. MVM is the most widely used operation in neural networks. The memristors are configured in a crossbar configuration where the inputs are applied as voltage signals, and the weights are programmed as the conductance of the memristors acts as an MVM operator,

and the resulting current vector is the product of the applied voltage vector and the conductance matrix.

3.4.1 Energy Calculation

The energy dissipated by a resistor with conductance g when a voltage v is applied across its terminals for a duration t is given by:

$$e = gv^2t. \tag{3.23}$$

Extending this calculation to a crossbar configuration, where the conductance matrix is represented by $\mathbf{G} \in \mathbb{R}^{m \times n}$, the voltage input vector by $\mathbf{V} \in \mathbb{R}^{n \times 1}$, and the inference time by t, the total energy dissipated by the crossbar during a MVM operation is expressed as:

$$E = \|\mathbf{G}\mathbf{V}^2\|_1 t, \tag{3.24}$$

where $\mathbf{V}^2$ indicates element-wise squaring of the vector $\mathbf{V}$, and $\|\cdot\|_1$ denotes the sum of the absolute values of the elements in the resulting vector from $\mathbf{G}\mathbf{V}^2$.

Using Equation (3.24), the energy consumed by an M-layer neural network during a single forward pass can be estimated as:

$$E_{\text{network}} = \sum_{k=1}^{M} \|\mathbf{G}_k \mathbf{V}_k^2\|_1 t, \tag{3.25}$$

where k represents the index of the layer out of a total of M layers in the network, $\mathbf{G}_k$ is the conductance matrix of the k-th layer, and $\mathbf{V}_k$ is the input voltage vector to the k-th layer.

To calculate the total energy consumed during the inference of a dataset with N samples, this energy is accumulated as follows:

$$E_{\text{total}} = \sum_{i=1}^{N} E_{\text{network}}^{(i)}, \tag{3.26}$$

where $E_{\text{network}}^{(i)}$ denotes the energy consumed for the i-th sample.

An upper bound on the energy consumption can be calculated by assuming that all weights are set to the maximum conductance $g_{\max}$ and all input voltages are at maximum voltage $v_{\max}$. The upper bound is given by:

$$E_{\max} = (\text{number of parameters}) \times g_{\max} \times v_{\max}^2 \times t, \tag{3.27}$$

where the number of parameters refers to the total number of parameters across all layers, and t is the time taken for a single inference.

3.4.2 Numerical Results

For the network in Table 3.2, we can calculate an upper bound for the power consumed by the analog neural network with $33,285$ parameters, a maximum input voltage $v_{\max} = 0.3$ V, the maximum conductance $g_{\max} = 25$ mS and an inference time for one forward pass of $t = 250$ ns, from Equation (3.27) to be 1.12 uW, which is 7 orders of magnitude less than the power consumed by a 15W Raspberry Pi Model 4B.

These results highlight the significant energy efficiency advantages of using PCM devices in analog neural networks. The substantial reduction in power consumption makes these networks highly suitable for deployment in energy-constrained environments. Furthermore, the combination of fast inference times and low energy requirements positions analog neural networks as a promising solution for real-time 5G network traffic prediction and other time-sensitive applications.

It is important to note that these calculations account only for the energy consumed by the memristor crossbars and does not include the power required by the input DACs, output ADCs, or any scaling amplifiers present in the input and output stages of the layers. Additionally, this calculation assumes that the crossbar is sufficiently large to accommodate an entire layer of the network, eliminating the need for tiling or partitioning the layer across multiple crossbars [57]. Such considerations are outside the scope of this work.

3.5 Chapter Summary

This chapter examines the prediction of cellular network traffic using memristive neural networks, highlighting its importance for optimizing telecommunications infrastructure. Accurate traffic forecasting is essential for capacity planning, load balancing, resource allocation, and maintaining high QoS, especially with emerging technologies like 5G and IoT. The study utilizes an unlabeled LTE dataset to evaluate network traffic predictions. The primary goal is to assess the performance of neural networks with analog computations by incorporating noise into network layers during inference.

We experiment with three robust training methods — incorporation of PCM noise, multiplicative log-normal noise, and additive Gaussian noise — to makeup for the performance loss due to memristor non-idealities. The results show that robust using the additive Gaussian noise significantly improves prediction accuracy over extended periods by enhancing the model's resilience to conductance decay due to drift. In addition, the chapter discusses estimation of energy consumption for deploying neural networks on analog hardware, which is crucial for battery-powered devices.

Chapter 4

Multi-Sensor Inference with Neural Networks Using Memristor-Based Analog Computing

Integrating DNNs with IoT devices has the potential to revolutionize computational intelligence. This integration enhances inference and classification capabilities in a broad spectrum of problems, including natural language processing, image recognition, and recommendation systems. However, the substantial computational demands of DNNs often exceed the processing capabilities of IoT sensors, posing a significant challenge in energy-limited scenarios.

Deploying DNNs on edge devices, such as sensors, is impractical due to their substantial computational demands, which conflict with the energy and latency limitations of these devices. Although network compression techniques, such as weight pruning and quantization [2], provide a means of deployment on edge devices, they significantly compromise performance. To address this challenge, [58] introduces a novel network splitting strategy through the implementation of an end-to-end architecture called Bottlenet++. In this setup, the encoder and

decoder function collectively as a machine learning-based joint source-channel coder. Meanwhile, [59] studies the problem of wireless image retrieval, focusing on situations where an edge device or sensor captures an object's image in raw format. The objective is to transmit a compact-dimensional signature of this image, aiding in the retrieval of similar images that represent the original object from a distinct dataset.

In most of the object identification setups, multiple sensors are spatially distributed around the object, capturing it from various angles to enrich the data with multiple perspectives, which requires data fusion from different resources. By integrating these observations through DNN, the system effectively overcomes the limitations of single-sensor data, offering a robust method for identifying the object.

Ref. [8] develops a transmission-efficient sensor fusion methodology with OTA aggregation for multi-sensor data. This method divides the DNN architecture into two components: a sensor-focused front-end for initial data collection and feature extraction, and a back-end for feature aggregation using the L_p-norm to complete the inference. The proposed approach approximates the maximum operation for sensor fusion in a transmission-efficient way, incorporating transformation invariance in classification tasks to facilitate precise object classification through OTA sensor fusion. This work was extended in [9], introducing a learnable L_p-norm to further optimize the fusion process.

In this chapter, we build upon the foundation laid in [8, 9] by exploring the use of analog memristor neural networks for multi-sensor fusion. Our approach addresses the challenges of computational demands and energy constraints in IoT edge devices. By integrating DNNs with IoT through in-memory analog computing using memristive crossbars, we propose a robust solution for complex computational tasks in distributed settings. This integration enhances energy efficiency while maintaining performance, representing a step forward in the development of more sustainable edge computing environments.

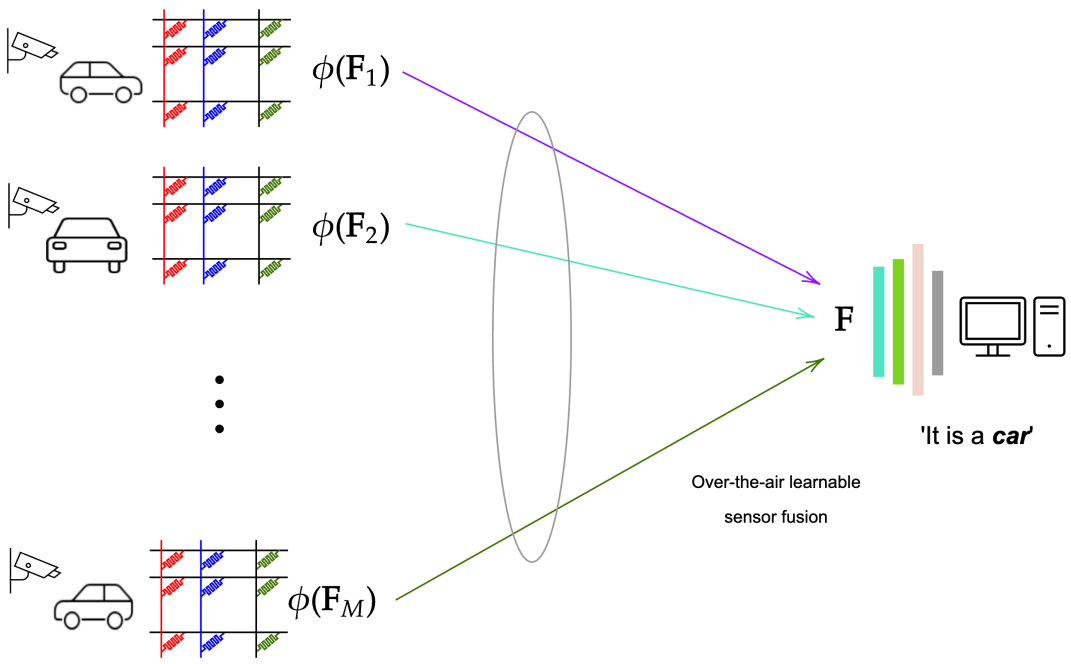

Figure 4.1: System model for the multi-sensor wireless inference with analog memristor neural networks.

The chapter is organized as follows: Section 4.1 outlines the system model. Section 4.2 describes the proposed application of wireless inference using in-memory computing. Numerical examples are presented in Section 4.3, and the chapter is concluded with a short summary in Section 4.4.

4.1 System Model

We consider a wireless sensor network with M analog sensors that observe the same phenomena for a common object. With the help of the multi-sensor structure, an inherent data augmentation is introduced to increase the inference reliability.

Our approach draws inspiration from a sensor fusion method presented in [8], which employs an L_p-norm inspired fusion technique. In [9], a learnable parameter, p, is introduced into the preprocessing function $\phi(\cdot)$, allowing the

optimization of the sensor fusion method to better suit the characteristics of the network and sensor distribution. We adopt this approach from [8, 9] and introduce a more intricate network architecture utilizing memristors for sensor devices. This integration provides improved energy efficiency without significantly sacrificing performance.

Memristors are good candidates for in-memory computing because they co-locate storage and computing by harnessing the non-volatility property of memory arrays. They can perform many concurrent multiply-and-accumulate operations in a single step. To simulate memristive behavior for computation in an analog neural network, multiple models are presented in the literature [16]. We base our results on a PCM device model presented in [34] and [60] due to its ease of implementation and foundation on empirical measurements from real device measurements. Further details on the model dynamics are given in Section 4.2.

4.2 Wireless Inference using In-memory Computing

In this section, we briefly describe the learnable L_p-norm inspired OTA sensor fusion technique from [8, 9]. As seen in Figure 4.1, each analog sensor $m \in \{1, \cdots, M\}$ processes its own detected raw data using a front-end network that uses in-memory computation made possible by a memristive crossbar chip to obtain the intermediate feature vector, $\mathbf{F}_m$. To complete the inference, these intermediate features are transmitted to a processing device utilizing an AWGN multiple access channel (MAC). Using the superposition property of the MAC channel, these transmissions take place simultaneously in a shared time/frequency slot, enabling OTA transmission. This method guarantees the effectiveness of the transmission. Here, sensor m transmits its intermediate feature vector $\mathbf{F}_m$ as follows:

$$\mathbf{x}_m = \phi(\mathbf{F}_m) = \mathbf{F}_m^p, \tag{4.1}$$

where $\mathbf{F}_m$ denotes the intermediate feature vector. Note that this intermediate feature vector might be either a 1D vector or a 2D matrix or 3D tensor, depending on the network properties. For ease of illustration, we assume $\mathbf{F}_m \in \mathbb{R}^d$.

Utilizing OTA transmission, the received signal can be written as

$$\mathbf{y} = \sum_{m=1}^{M} \mathbf{x}_m + \mathbf{n} = \sum_{m=1}^{M} \mathbf{F}_m^p + \mathbf{n}, \tag{4.2}$$

using the superposition property of MAC, in which $\mathbf{y} \in \mathbb{R}^d$, and $\mathbf{n} \in \mathbb{R}^d$ is the AWGN noise with variance σ_n^2.

At the central processing device, the i-th element of the received signal will be processed as

$$y(i)^{\frac{1}{p}} = \left(\sum_{m=1}^{M} F_m(i)^p + n(i) \right)^{\frac{1}{p}}, \tag{4.3}$$

which is then used to complete the inference.

Note that this pre-processing function is the same as described in [8] for a constant $p \geq 0$. However, we are able to capture a spectrum of behaviors using the trainable parameter p as introduced in [9]. To be more precise, the function approximates the maximum operation for sufficiently large values of p, but for $p = 1$, it corresponds to the summation of features. Depending on the environment, sensor, and network structure, this parameter can converge to an ideal value during training that maximizes total inference accuracy.

The previously described model ensures the transmission efficiency of a multi-sensor network by utilizing a trainable OTA sensor fusion approach. In addition to this improvement, the energy efficiency of the overall system can be further enhanced by employing memristors for neural network operations. Note that in-memory computing using memristors can be applied to both the front-end and back-end of a neural network. However, in this study, we focus solely on the front-end network, specifically analog sensors with memristive neural networks, while the proposed approach can easily be generalized to full analog networks.

It is important to note that the use of memristors may introduce additional impairments during the process of obtaining intermediate features due to imperfect and noisy operations. To accurately model memristive computations, we utilize the PCM inference noise model described in Section 2.4. This model identifies three types of noise elements: programming noise, drift noise, and read noise, which occur during weight updates and readings, as well as a time-dependent decay of programmed values.

Programming noise arises from discrepancies between target and actual conductance values programmed in the hardware. Drift noise reflects the time-dependent decay of programmed conductance due to the structural relaxation of the phase-change material. Read noise represents instantaneous fluctuations in conductance during matrix-vector multiplications due to intrinsic noise characteristics of PCM devices.

To translate the trained weight values into conductance terms, a differential configuration employing a pair of memristors is used. This process, along with the handling of noise effects during conductance mapping, is described in detail in Section 2.4.

4.3 Numerical Results

As described in Section 4.1, we consider a multi-sensor distributed wireless inference setup [8, 9] with M sensors, in which L_p-norm inspired sensor fusion is used for OTA aggregation. For the described setup, training is performed offline, and the weights are shared with the sensors and the central device for real-time inference.

During inference, to simulate in-memory computation using PCM type memristors, we replicate and map the sensor weights to all M sensors as described in Section 4.2. We add the memristor noise to the network weights during the

Table 4.1: Network architecture for MNIST classification.

Front-end (sensors)	Image: 28×28
	5×5 convolutional layer, 10 channels, ReLu, stride: $(1, 1)$
Back-end (device)	Fully connected layer $(5760, 50)$, ReLu
	Output layer: fully connected $(50, 10)$

forward pass and compute a noisy activation for the m-th sensor as

$$\mathbf{F}_m = f\left(w', \mathbf{u}_m\right), \tag{4.4}$$

where w' are the network weights after adding noise as described in Equation (2.31) and $\mathbf{u}_m$ is the input for the m-th sensor. Unlike offline training, the inference phase is performed in real time, and uses a pre-trained neural network by employing the forward pass on analog memristive sensors. This allows us to simulate a real scenario in which each sensor will undergo a different weight distribution after programming.

4.3.1 MNIST

In our initial investigation, we experiment with the MNIST [14] classification, utilizing the network architecture given in Table 4.1. We train the network under AWGN noise conditions, with SNR values set at $\{-5, 0, 10\}$ dB and sensors ranging from $M = 1$ to $M = 10$. To simulate different views for each sensor, we rotate each image randomly between $0 - 180$ degrees. Each configuration is trained with Adam optimizer [61] at a learning rate of $\eta = 0.001$ for 200 epochs. The best model is saved for subsequent analog inference. Figure 4.2 shows the test accuracy for each scenario. Each data point represents the mean of 25 inferences, each derived from five programming trials, further analyzed over the communication channel five times, with shaded areas indicating the standard deviation of these trials.

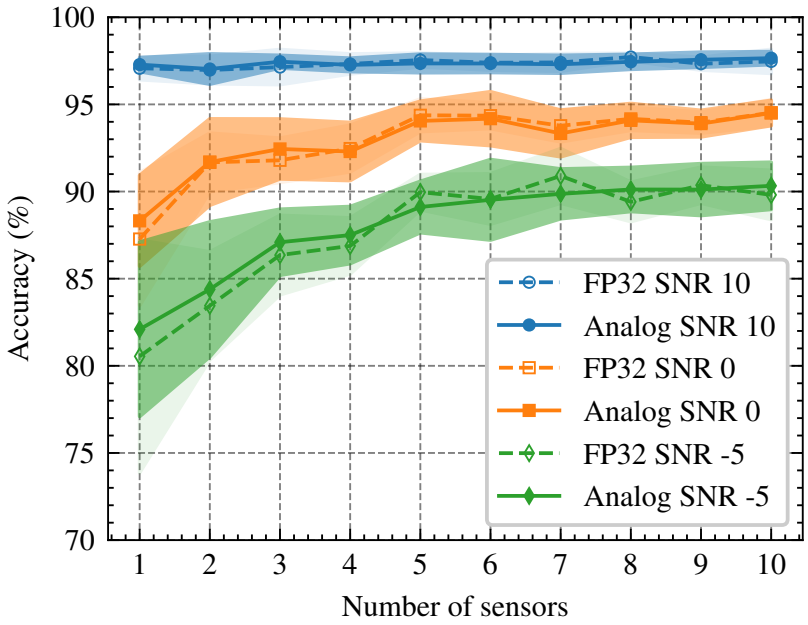

Figure 4.2: Test accuracy for MNIST classification with digital and analog sensors against number of sensors.

The analog sensor scenario exhibits performance that closely matches the digital implementation (FP32), thus supporting the advantages of integrating analog devices into neural networks with minimal performance degradation. The variability in the results, measured by the standard deviation, is attributable to two primary factors: first, the programming and read noise inherent in the memristive devices and second, the fluctuations in the wireless channel. An increase in the number of sensors and an improvement in the channel SNR contribute to more predictable performance with a significant reduction in standard deviation.

4.3.2 ModelNet

For our subsequent analysis, we selected the ModelNet [62] dataset, with the network architecture given in 4.2. This data set is significantly more complex, comprising 40 distinct objects, each represented by 12 unique 2D views of the corresponding 3D model. The network performance is evaluated for three different scenarios of training and test channels 1) Perfect channel state information

Table 4.2: MVCNN network architecture for ModelNet classification.

Front-end (sensors)	Image: $224 \times 224 \times 3$
	3×3 convolutional layer, 32 channels. ReLU, stride:4, padding: 2
	2D maxpooling with kernelsize=3, stride=2
	Dropout with probability 0.5
Back-end (device)	Fully connected layer $(23328, 8196)$ ReLU
	Dropout with probability 0.5
	Fully connected layer $(8196, 2048)$ ReLU
	Output layer: fully connected layer $(2048, 40)$

(CSI), 2) CSI mismatch and, 3) Robust training. Each configuration is trained with Adam optimizer, with a learning rate set at $\eta = 0.0001$, for 100 epochs, incorporating configurations of 1, 5, and 12 sensors. Each model is tested for channels with SNRs set to $\{0, 5, 10\}$ dB.

Perfect CSI: Figure 4.3 presents the test accuracy outcomes for analog and digital sensors under conditions where the SNR for both testing and training phases is identical. As per expectation, increasing the number of sensors improves the performance for all channel conditions. Moreover, an observed decrement of 3-5% in accuracy is seen when employing analog sensors. This phenomenon is attributed to the increased complexity of the network, which in turn introduces a greater chance for errors within the network weights as each weight in the sensor layers is subjected to a weight-dependent noise as defined in Equation (2.17).

CSI mismatch: Figures 4.4 and 4.5 display the test accuracy results under various SNR conditions where the models are tested at an SNR different than the training conditions. The results clearly indicate that the best performance is achieved when the training and test channel conditions are matched, while a significant loss in performance is observed when there is a mismatch. Specifically, the model trained at 10 dB SNR and 12 sensors achieved an accuracy of 84%

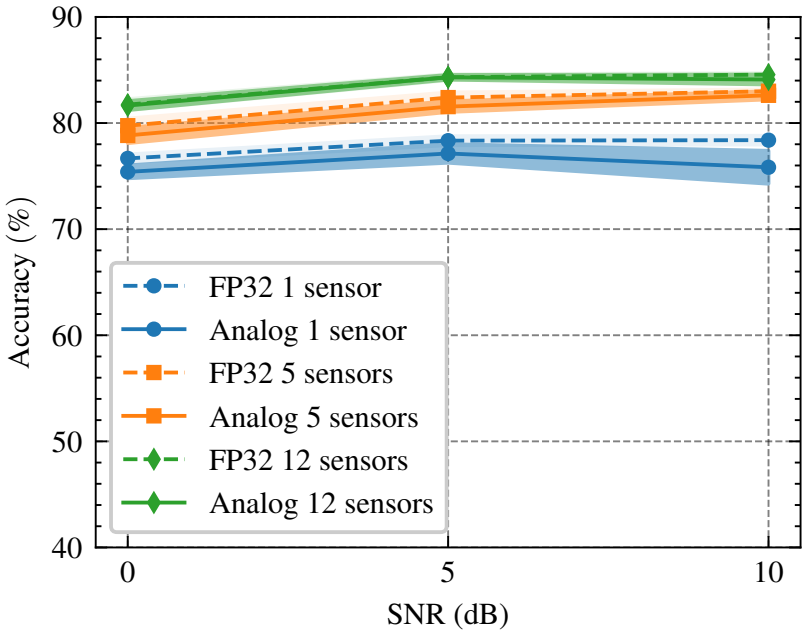

Figure 4.3: Test accuracy for ModelNet classification utilizing digital and analog sensors, with identical SNR during training.

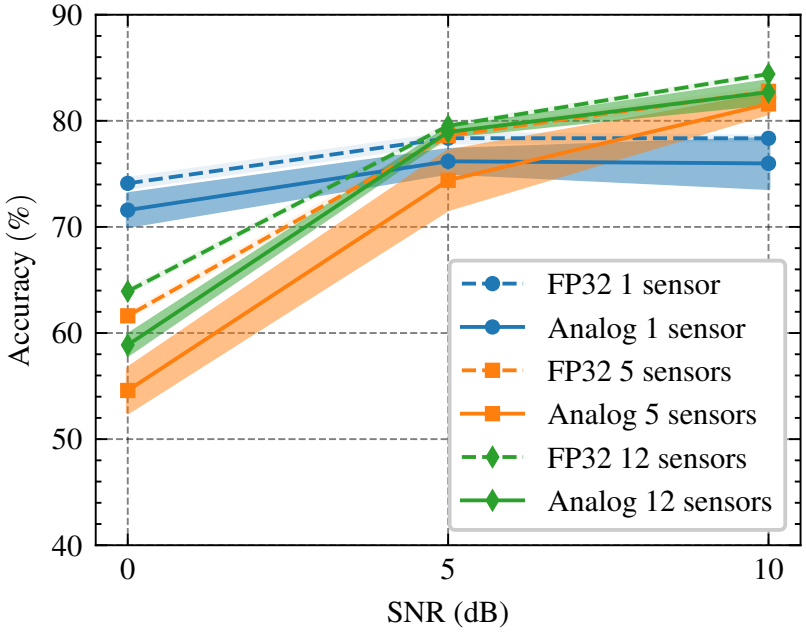

Figure 4.4: Test accuracy for ModelNet classification utilizing digital and analog sensors, with 10dB SNR during training.

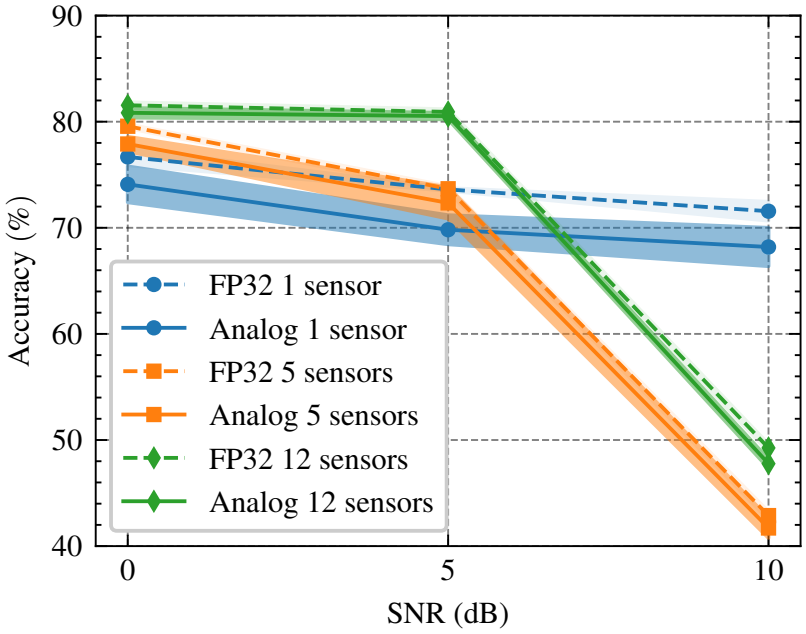

Figure 4.5: Test accuracy for ModelNet classification utilizing digital and analog sensors, with 0dB SNR during training.

when tested at 10 dB, but the accuracy dropped to 78% and 64% when tested at 5 dB and 0 dB, respectively. Similarly, the model trained at 0 dB SNR showed a peak accuracy of 82% at 0 dB but decreased to 80% and 49% at 5 dB and 10 dB, respectively. This strengthens our findings from Figure 4.3 that incorporating CSI information during the training phase significantly enhances the network's performance by making it robust to a range of channel conditions. The numerical results provide strong evidence that matching the training and testing channel conditions is crucial for optimal performance. Similar observations can be made for both the five-sensor and single-sensor models, where accuracy peaks when the training and test channel SNR match. In a practical setup, achieving good performance across a range of channel conditions would require training a model for each possible channel condition, which is not feasible for practical reasons.

Robust training: To make the setup more practical by avoiding the need to train multiple models for various channel conditions, we propose a training approach where each epoch employs a randomly selected SNR between 0 dB and 10 dB, chosen uniformly on a linear scale. Testing is then performed under

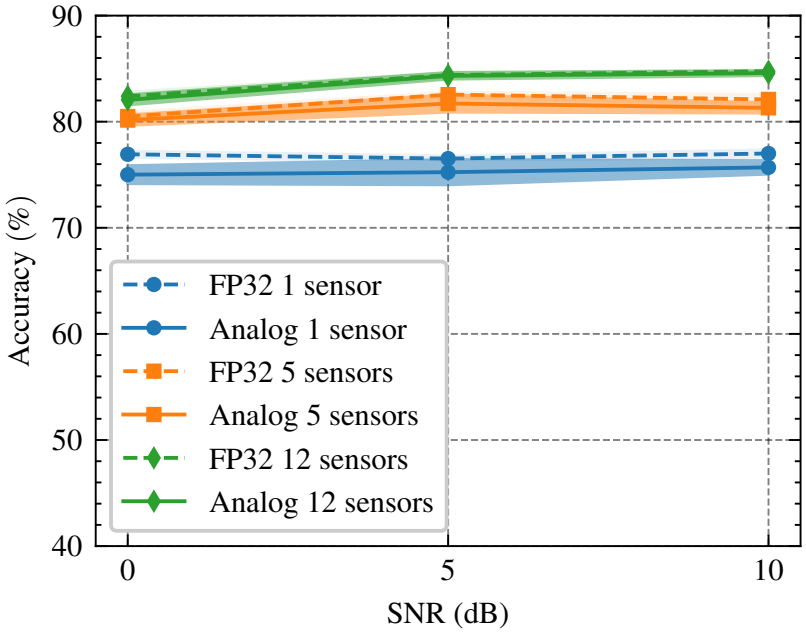

Figure 4.6: Test accuracy for ModelNet classification utilizing digital and analog sensors, with a random SNR during training.

various SNR conditions, with results shown in Figure 4.6. Each data point in the results represents the mean of 25 inferences, each derived from five distinct programming trials, with each trial inferred five times over the communication channel. The shaded regions in the figures indicate the standard deviation of these trials, providing a clear depiction of the variability and reliability of our results. Notably, training with a random SNR per epoch increases the network's adaptability to fluctuating channel conditions during testing, resulting in a flatter response across a range of SNRs. This method enhances the robustness of the model, ensuring reliable performance even when channel conditions vary unpredictably. By incorporating this randomized training approach, we address the impracticality of training separate models for each potential channel condition, thus offering a more feasible and efficient solution for real-world applications.

Figure 4.7 presents the test accuracy results for digital and analog sensors with drift noise. The model is trained using five sensors and an SNR uniformly selected between 0 dB and 10 dB. Subsequently, it is tested under a 10 dB channel SNR with drift noise given in Equation (2.19) that progressively decays the weights

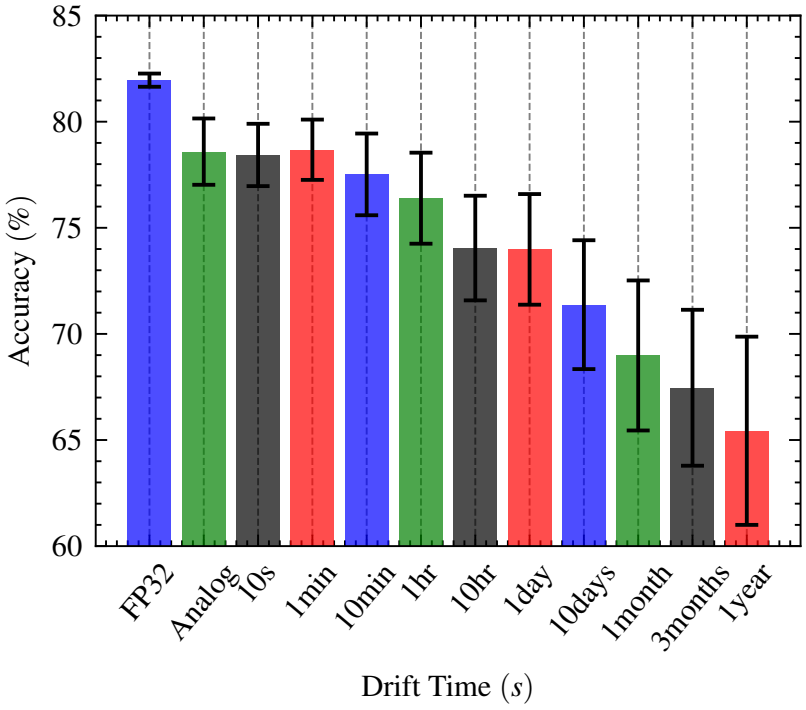

Figure 4.7: Accuracy vs drift time for the network from Table 4.2 trained with 5 sensors at 10 dB SNR.

over time. The model is evaluated at various time intervals: 10 seconds, 1 minute, 10 minutes, 1 hour, 10 hours, 1 day, 10 days, 1 month, 3 months, and 1 year. Each bar in Figure 4.7 represents an average of 1000 runs with 100 programming or drift trials, and each trial is further inferred over the communication channel 10 times. The error bars indicate the standard deviation. This setup models the long-term impact of weight drift in sensors employing PCM-type devices in memristive neural networks. After converting to analog, we observe an initial accuracy drop of 4-5%, which gradually decreases to just above 65% over the course of one year. Our extensive experimental design and consistent results across 1000 iterations provide a statistically sound understanding of how memristive neural networks degrade over time due to PCM conductance drift. This gradual decline in accuracy mirrors real-world scenarios and offers valuable data for predicting the reliability of these networks in practical applications. The clear pattern of accuracy decay underscores the need for continuous monitoring and reprogramming in systems using these sensors to ensure sustained performance. This study

identifies the reprogramming interval to maintain an acceptable classification accuracy.

4.4 Chapter Summary

This chapter focuses on enhancing the energy efficiency of IoT devices by integrating multi-sensor fusion with analog memristor neural networks. We incorporate a learnable L_p-norm inspired approach for OTA aggregation of outputs from spatially distributed sensors as introduced in [8, 9]. This method introduces an innovative OTA transmission mechanism, combining outputs directly over a multiple access AWGN channel. As a novel contribution, we incorporate analog computations facilitated by memristors for multi-sesnor distributed inference. Memristors, known for their energy-efficient in-memory computing capabilities, form the foundation of our approach, in which the computations are performed in an analog domain. This approach significantly reduces energy consumption and computational overhead in edge computing applications. By addressing the computational and energy constraints of DNNs on edge devices, this dual focus on memristors and L_p-norm optimization promotes more sustainable and energy efficient computational paradigms and transmission efficiency. Our results demonstrate substantial improvements in energy efficiency, highlighting the potential of this approach for future IoT applications and distributed systems.

Chapter 5

Neural Successive Cancellation Decoder for Polar Codes Using Analog In-Memory Computing

Polar codes represent a significant advancement in channel coding, capable of achieving the Shannon capacity, which makes them a vital component in next-generation communication systems [63]. Their ability to classify bit channels as either good or bad based on polarization enables efficient error correction, particularly as the code length increases. The original decoder for polar codes, the succesive cancellation (SC) decoder, introduced by Arikan, is a fundamental method but inherently sequential, leading to relatively high latency, especially in real-time applications.

In recent years, there has been a growing interest in integrating DNNs with polar code decoders to leverage the parallel processing capabilities of neural networks and mitigate the limitations of the SC decoder [64]. However, traditional implementations still rely on von Neumann architectures, where the separation of memory and computation units can result in inefficiencies in power consumption and data transfer latency.

This chapter explores a more efficient approach to polar code decoding by incorporating memristor-based in-memory computing within the neural successive cancellation (NSC) decoder. Memristors, known for their low-latency and energy-efficient capabilities, are utilized to address the bottlenecks of traditional hardware architectures, offering a potentially efficient solution for decoding polar codes. The integration of memristors within the NSC decoder may facilitate real-time, low-power decoding, which is particularly relevant for edge devices with limited computational resources.

5.1 Neural Successive Cancellation Decoding

SC decoding is the original algorithm introduced by Arikan for polar codes. It operates by sequentially estimating each bit of the codeword, starting from the most significant bit to the least significant one. The decoder leverages the channel polarization property, classifying bit channels as either "good" or "bad". The SC decoder computes log-likelihood ratios (LLRs) for the received signal and makes hard decisions based on these values, with each bit's estimation depending on the decisions made for all previous bits. The SC decoder, illustrated in Figure 5.1, decodes bit channels sequentially, which can be a limitation in scenarios that require low latency. This serial nature results in high latency, particularly for long codewords, and may limit the SC decoder's suitability for parallel processing hardware like GPUs.

To address these limitations, researchers have integrated ANNs with the SC decoder, resulting in the NSC decoder [64]. The NSC decoder aims to exploit the parallelization capabilities of ANNs to speed up the decoding process while retaining the structural benefits of the SC decoder.

NSC Decoding: NSC decoding is an enhancement of the SC algorithm that integrates ANNs into the decoding process. Unlike SC decoding, which operates serially, NSC decoding introduces parallelism by embedding ANNs at specific stages within the SC decoding tree. These ANNs are trained to predict bit values

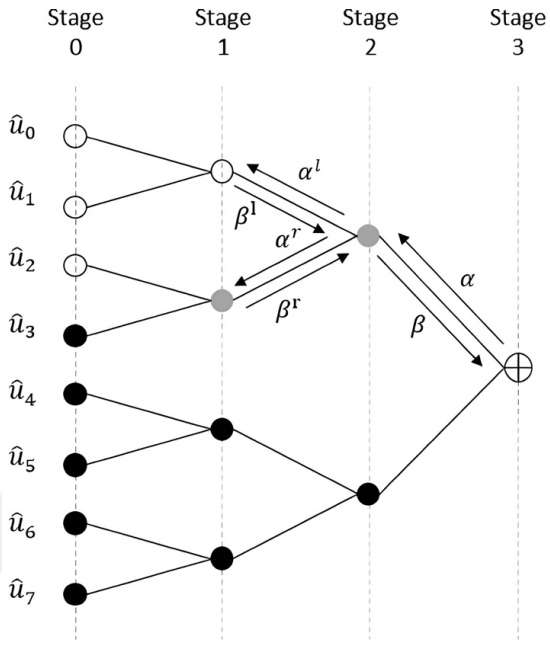

Figure 5.1: Binary tree of the SC decoder for $\mathcal{P}(8,3)$.[64]

based on LLRs, allowing the NSC decoder to process multiple bits simultaneously at certain stages. This significantly reduces overall latency while maintaining the reliability of the original SC algorithm. The NSC decoder strikes a balance between the serial structure of SC and the parallel processing strengths of ANNs, making it suitable for low-latency applications.

The NSC decoder operates by incorporating ANNs at specific stages within the SC decoding process, where parallel processing can effectively reduce latency. As illustrated in Figure 5.2, ANNs are placed within the SC decoding tree, trained to estimate message bits based on the LLR values computed by the SC decoder. The strategic placement of ANNs allows the NSC decoder to balance the serial nature of the SC decoder with the parallel processing strengths of ANNs.

In the NSC decoder, multiple neural networks are trained independently, each corresponding to different stages of the SC decoding tree. This architecture provides flexibility, allowing the placement of ANNs at various stages depending on the specific requirements of the code length and performance goals. As the

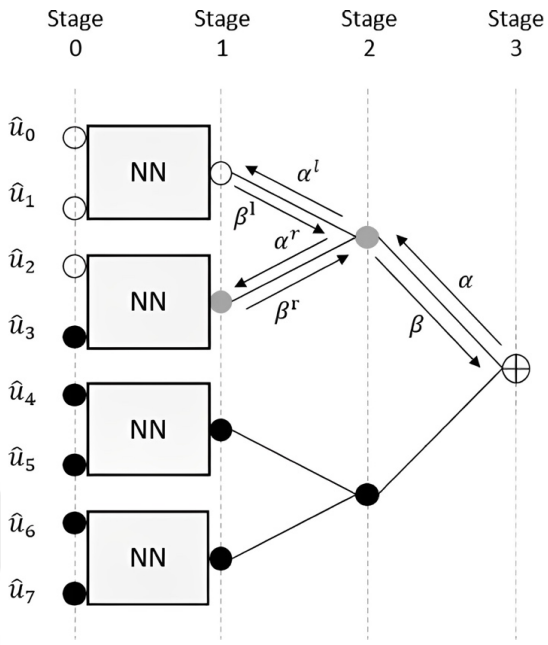

Figure 5.2: Binary tree of the NSC decoder for $\mathcal{P}(8, 3)$.[64]

stage number increases, the performance of the NSC decoder increasingly resembles that of a traditional SC decoder due to the greater reliance on SC decoding stages.

5.1.1 Memristor-Based NSC

To enhance the efficiency of the NSC decoder further, we propose the use of memristor-based in-memory computing. By replacing traditional neural network layers with memristor crossbars, the NSC decoder can perform matrix-vector multiplications directly within the memory, thereby reducing data movement and potentially increasing computational speed. The integration of memristors in the NSC decoder could facilitate low-latency and energy-efficient decoding, making it a viable solution for real-time applications in power-constrained environments [65].

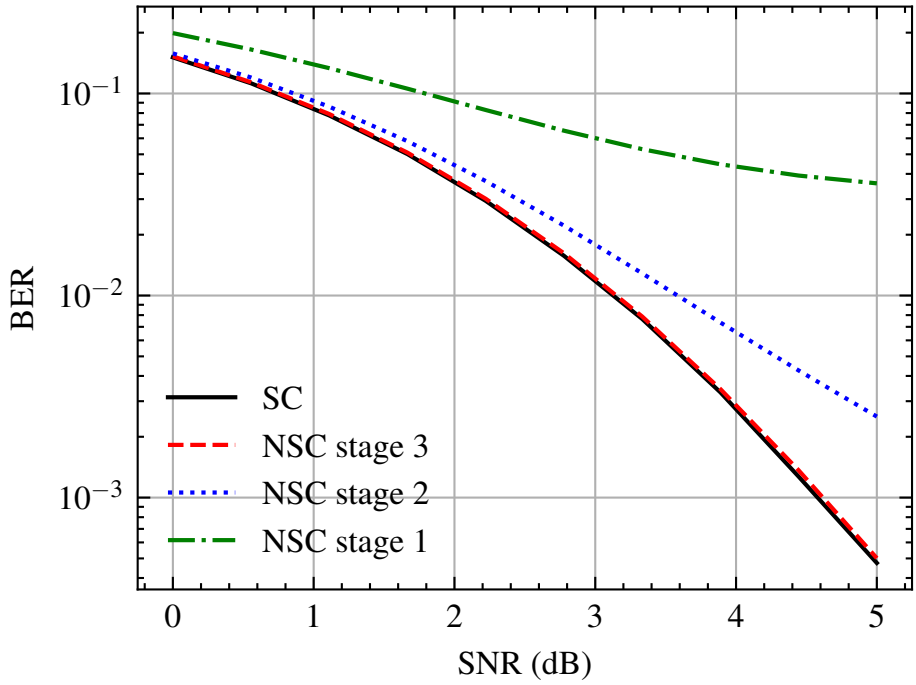

Figure 5.3: $\mathcal{P}(32, 16)$ polar code NSC decoder – BER vs. SNR comparative performance with NNs placed at different stages.

5.2 Numerical Examples

In this section, we compare the decoding performance of polar codes using SC, NSC, and analog NSC decoders. Figure 5.3 illustrates the BER performance for $\mathcal{P}(32, 16)$, showing how the NSC decoder performs when neural networks are incorporated at different stages of the SC algorithm. As the stage number increases from stage 1 to stage 3, the NSC decoder's performance increasingly aligns with that of the traditional SC decoder. This improvement is due to the larger portion of the decoding process being handled by SC decoding, which reduces the reliance on neural networks.

Figure 5.4 compares the BER performance for a $\mathcal{P}(64, 32)$ polar code using the SC and NSC decoders. At stage 3, the NSC decoder performs comparably to the SC decoder, with a performance margin within 0.5 dB. The advantage of the NSC decoder is its faster decoding speed, achieved through the parallelism enabled by incorporating ANNs into the decoding process.

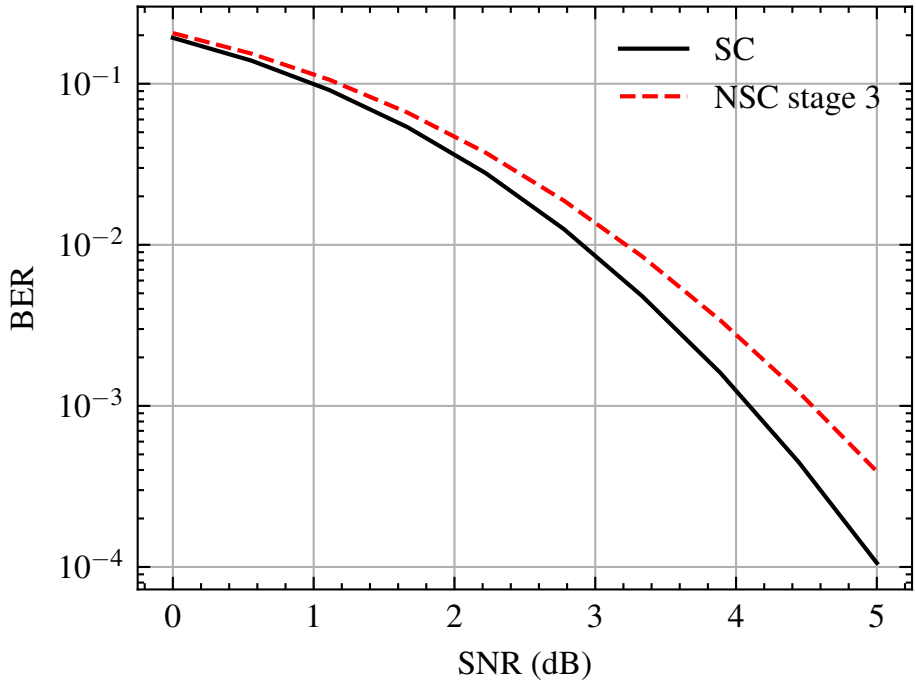

Figure 5.4: $\mathcal{P}(64, 32)$ polar code – comparison of SC and NSC decoding results.

Figure 5.5 presents the scenario where the neural network layers of the NSC decoder are implemented using analog memristive devices. The PCM noise model is applied to simulate the analog computations. Due to the inherent noise in PCM devices, the analog NSC decoder exhibits a higher BER compared to the digital implementation. This increase in BER is primarily attributed to programming noise and drift in the network weights over time. Nevertheless, without the effect of drift noise (i.e., at $t = 0.0\,\mathrm{s}$), the analog NSC decoder only loses 0.5 dB in performance compared to the digital NSC decoder. Despite the performance loss, the analog NSC decoder provides significant benefits in terms of real-time processing and energy efficiency, making it a viable solution for edge devices with limited power resources.

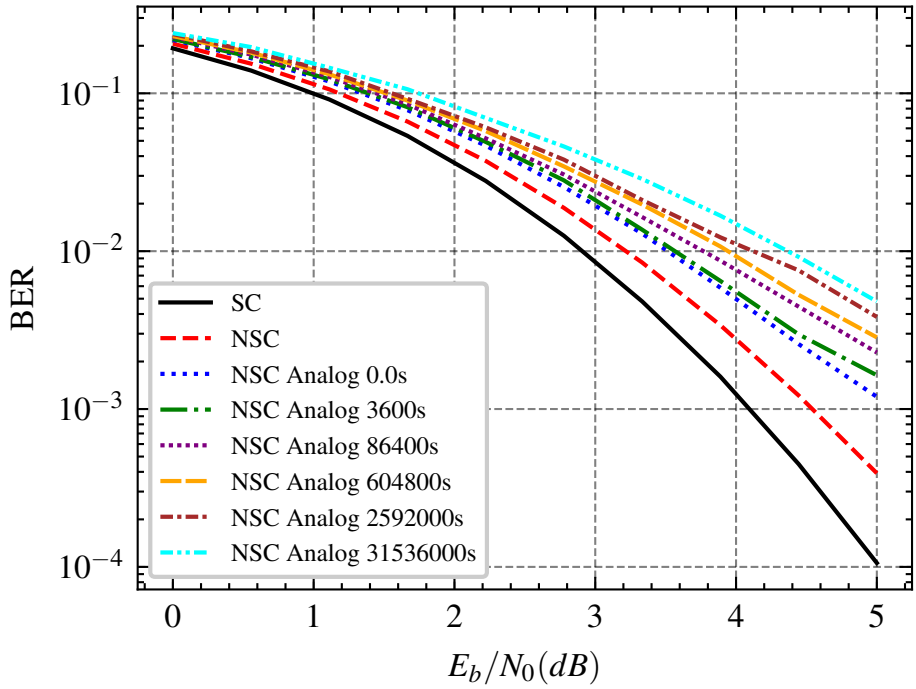

Figure 5.5: $\mathcal{P}(64, 32)$ polar code – analog NSC decoding results after weight drift.

5.3 Chapter Summary

This chapter presented the NSC decoder for polar codes using analog in-memory computing, focusing on reducing the computational and power costs associated with deploying DNNs on edge devices. By integrating memristors within the NSC decoder, the approach eliminates the need for data transfer between memory and processor units, thereby reducing latency and power consumption. The NSC decoder combines the benefits of SC decoders with the parallel processing capabilities of ANNs, resulting in a highly efficient and low-power solution for polar code decoding. The performance evaluation demonstrated that the NSC decoder achieves comparable BER performance to traditional SC decoders, with minimal performance loss even when implemented with analog memristive neural networks. This makes the analog NSC decoder particularly suitable for real-time decoding in power-constrained environments.

Chapter 6

Conclusions and Future Directions

In this thesis, we explored the integration of machine learning algorithms with in-memory analog computing to enhance the performance and efficiency of wireless communication systems. The study addresses the computational challenges posed by the increasing complexity of wireless applications and the limitations of traditional digital computing architectures. The research focuses on the performance bottlenecks and energy inefficiencies inherent in conventional digital computing architectures within wireless communication systems. The primary objective is to investigate the potential of integrating machine learning algorithms with in-memory analog computing, particularly using memristor-based architectures, to improve computational efficiency and achieve energy savings at the edge of wireless networks.

The key findings of this research can be summarized as follows: Analog computations using memristive devices show promise for low-latency, energy-efficient network traffic prediction. While noise in analog computations degrades the prediction performance, the introduction of robust training techniques has demonstrated the potential to mitigate this degradation. Specifically, employing robust

training with AWGN has been shown to improve prediction accuracy over extended periods, enhancing resistance to conductance drift in memristors. Additionally, the deployment of deep neural networks on edge IoT devices for multi-sensor data fusion has been explored. The learnable L_p-norm inspired sensor fusion with analog in-memory computing appears promising for enabling efficient data fusion from multiple sensors while maintaining high accuracy and robustness, even in challenging scenarios. This approach could offer significant energy efficiency gains, which may be advantageous for battery-powered and energy-constrained IoT applications. Moreover, experiments with the analog NSC decoder for polar codes suggest potential benefits in reducing latency and energy consumption using memristors, potentially offering advantages over traditional computing architectures in specific real-time applications.

It is also important to acknowledge the limitations of this research. The performance of memristor-based analog computing is influenced by noise and variability in memristive devices. Although robust training techniques can be employed to mitigate these effects, further research is required to thoroughly understand and address these challenges. Additionally, the scalability of the proposed approaches to larger and more complex neural network architectures remains an open question. Future research should explore the scalability and practical implementation of these methods in large-scale deployments. Implementing the proposed memristor-based analog computing systems in real-world environments also presents several challenges. Integration with existing digital infrastructure, manufacturing variability, and the long-term stability of memristive devices are factors that require comprehensive investigation. Moreover, practical deployment in diverse application scenarios necessitates extensive testing to ensure robustness, reliability, and compatibility with current communication standards. Addressing these practical implementation challenges will be crucial for successfully translating research findings into commercial and industrial applications.

In light of the findings and limitations identified in this research, several recommendations for future studies are proposed. First, future research should focus on developing advanced noise mitigation strategies and robust training techniques to enhance the reliability and robustness of memristive neural networks. Second,

real-world implementations and extensive testing of the proposed approaches across various application domains, such as telecommunications, the IoT, and autonomous systems, are essential to validate their practical effectiveness and identify areas for further improvement. Third, the development of more realistic and high-fidelity simulations for memristive crossbar-based neural networks is necessary to better model real-world scenarios. Finally, improvements in energy consumption calculations should consider the crossbar peripheral circuitry, thereby providing a more comprehensive assessment of the system's efficiency.

This research demonstrates the potential of integrating machine learning with in-memory analog computing using memristors. While significant challenges remain in achieving computational efficiency and reducing energy consumption, there is considerable scope for future work to build on these findings and further enhance the capabilities and applications of memristive neural networks across various domains. Detailed simulations and hardware implementations will be particularly important for advancing this research area.

Bibliography

[1] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: Wireless communication meets machine learning," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[2] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 129–146, 2020.

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

[4] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[5] B. Isik, K. Choi, X. Zheng, T. Weissman, S. Ermon, H.-S. P. Wong, and A. Alaghi, "Neural network compression for noisy storage devices," 2023.

[6] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, pp. 529–544, July 2020.

[7] D. J. Kösters, B. A. Kortman, I. Boybat, E. Ferro, S. Dolas, R. Ruiz de Austri, J. Kwisthout, H. Hilgenkamp, T. Rasing, H. Riel, A. Sebastian, S. Caron, and J. H. Mentink, "Benchmarking energy consumption and latency for neuromorphic computing in condensed matter and particle physics," *APL Machine Learning*, vol. 1, no. 1, p. 016101, 2023.

[8] B. Tegin and T. M. Duman, "Transformation-invariant over-the-air combining for multi-sensor wireless inference," in *IEEE Global Communications Conference (GLOBECOM)*, (Kuala Lumpur, Malaysia), pp. 2937–2942, Dec. 2023.

[9] B. Tegin, *Federated learning and distributed inference over wireless channels*. PhD thesis, Bilkent University, 2023.

[10] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, June 2017.

[11] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[12] M. N. Bojnordi and E. Ipek, "Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–13, IEEE, Mar. 2016.

[13] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[14] Y. Lecun, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, 1998.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (Las Vegas, NV, USA), pp. 770–778, IEEE, June 2016.

[16] J. P. Carbajal, J. Dambre, M. Hermans, and B. Schrauwen, "Memristor models for machine learning," *Neural Computation*, vol. 27, no. 3, pp. 725–747, 2015.

[17] J. P. Strachan, A. C. Torrezan, F. Miao, M. D. Pickett, J. J. Yang, W. Yi, G. Medeiros-Ribeiro, and R. S. Williams, "State dynamics and modeling of tantalum oxide memristors," *IEEE Transactions on Electron Devices*, vol. 60, no. 7, pp. 2194–2202, 2013.

[18] T. Gokmen and W. Haensch, "Algorithm for training neural networks on resistive device arrays," *Frontiers in Neuroscience*, vol. 14, p. 103, 2020.

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

[20] M. A. Lastras-Montaño and K.-T. Cheng, "Resistive random-access memory based on ratioed memristors," *Nature Electronics*, vol. 1, pp. 466–472, Aug. 2018.

[21] J. Cui, F. An, J. Qian, Y. Wu, L. L. Sloan, S. Pidaparthy, J.-M. Zuo, and Q. Cao, "CMOS-compatible electrochemical synaptic transistor arrays for deep learning accelerators," *Nature Electronics*, vol. 6, pp. 292–300, Apr. 2023.

[22] Y. Li, S. Kim, X. Sun, P. Solomon, T. Gokmen, H. Tsai, S. Koswatta, Z. Ren, R. Mo, C. C. Yeh, W. Haensch, and E. Leobandung, "Capacitor-based cross-point array for analog neural network with record symmetry and linearity," in *IEEE Symposium on VLSI Technology*, (Honolulu, HI, USA), pp. 25–26, June 2018.

[23] K.-U. Demasius, A. Kirschen, and S. Parkin, "Energy-efficient memcapacitor devices for neuromorphic computing," *Nature Electronics*, vol. 4, no. 10, pp. 748–756, 2021.

[24] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, pp. 641–646, Jan. 2020.

[25] Z. Sun, G. Pedretti, A. Bricalli, and D. Ielmini, "One-step regression and classification with cross-point resistive memory arrays," *Science Advances*, vol. 6, no. 5, p. 2378, 2020.

[26] J.-W. Jang, B. Attarimashalkoubeh, A. Prakash, H. Hwang, and Y.-H. Jeong, "Scalable neuron circuit using conductive-bridge ram for pattern reconstructions," *IEEE Transactions on Electron Devices*, vol. 63, no. 6, pp. 2610–2613, 2016.

[27] Z. Sun, G. Pedretti, E. Ambrosi, A. Bricalli, W. Wang, and D. Ielmini, "Solving matrix equations in one step with cross-point resistive arrays," *Proceedings of the National Academy of Sciences*, vol. 116, pp. 4123–4128, Mar. 2019.

[28] J. Randrianantenaina, A. Baran, N. Korkmaz, and R. Kilic, "A new nonlinear ion drift model of memristor element and its versatile analog reconfigurable realizations," *Journal of Circuits, Systems and Computers*, 2023.

[29] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices," *Journal of Applied Physics*, vol. 106, no. 7, 2009.

[30] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Team: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 211–221, 2013.

[31] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[32] G. B. Beneventi, M. Ferro, and P. Fantini, "$1/f$ noise in 45-nm reset-state phase-change memory devices: Characterization, impact on memory readout operation, and scaling perspectives," *IEEE Electron Device Letters*, vol. 33, no. 11, pp. 1559–1561, 2012.

[33] D. Fugazza, D. Ielmini, S. Lavizzari, and A. L. Lacaita, "Random telegraph signal noise in phase change memory devices," in *IEEE International Reliability Physics Symposium*, (Anaheim, CA, USA), pp. 743–749, IEEE, May 2010.

[34] S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian, and E. Eleftheriou, "A phase-change memory model for neuromorphic computing," *Journal of Applied Physics*, vol. 124, p. 152135, Oct. 2018.

[35] S. Oh, Z. Huang, Y. Shi, and D. Kuzum, "The impact of resistance drift of phase change memory (PCM) synaptic devices on artificial neural network performance," *IEEE Electron Device Letters*, vol. 40, no. 8, pp. 1325–1328, 2019.

[36] L. Xia, B. Li, T. Tang, P. Gu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "Mnsim: Simulation platform for memristor-based neuromorphic computing system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 1009–1022, 2018.

[37] Z. Zhu, H. Sun, K. Qiu, L. Xia, G. Krishnan, G. Dai, D. Niu, X. Chen, X. S. Hu, Y. Cao, Y. Xie, Y. Wang, and H. Yang, "MNSIM 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, GLSVLSI '20, (New York, NY, USA), p. 83–88, Association for Computing Machinery, Sept. 2020.

[38] A. Lu, X. Peng, W. Li, H. Jiang, and S. Yu, "Neurosim simulator for compute-in-memory hardware accelerator: Validation and benchmark," *Frontiers in Artificial Intelligence*, vol. 4, 2021.

[39] M. Le Gallo, C. Lammie, J. Büchel, F. Carta, O. Fagbohungbe, C. Mackin, H. Tsai, V. Narayanan, A. Sebastian, K. El Maghraoui, *et al.*, "Using the IBM analog in-memory hardware acceleration kit for neural network training and inference," *APL Machine Learning*, vol. 1, no. 4, 2023.

[40] M. H. Amin, M. E. Elbtity, and R. Zand, "Imac-sim: A circuit-level simulator for in-memory analog computing architectures," in *Proceedings of the Great Lakes Symposium on VLSI*, (TN, Knoxville, USA), pp. 659–664, June 2023.

[41] Synopsys Inc., "PrimeSim SPICE simulator for analog, RF, and mixed-signal design." `https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-spice.html`.

[42] S. N. Laboratories, "CrossSim." https://cross-sim.sandia.gov, 2024.

[43] J. Ansel *et al.*, "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, (CA, La Jolla, USA), ACM, Apr. 2024.

[44] Ericsson, "Mobile data traffic forecast – Ericsson mobility report," tech. rep., Ericsson, 2023.

[45] W. Jiang, "Cellular traffic prediction with machine learning: A survey," *Expert Systems with Applications*, vol. 201, p. 117163, 2022.

[46] CTTC, "SUPERCOM - AI Challenge 2022." `https://supercom.cttc.es/index.php/supercom-events/events-past/itu-challenge-2022`.

[47] H. D. Trinh, A. F. Gambin, L. Giupponi, M. Rossi, and P. Dini, "Mobile traffic classification through physical control channel fingerprinting: A deep learning approach," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1946–1961, 2020.

[48] Y. Hua, Z. Zhao, Z. Liu, X. Chen, R. Li, and H. Zhang, "Traffic prediction based on random connectivity in deep learning with long short-term memory," in *IEEE 88th Vehicular Technology Conference (VTC-Fall)*, (Chicago, IL, USA), pp. 1–6, IEEE, Aug. 2018.

[49] Y. Shu, M. Yu, O. Yang, J. Liu, and H. Feng, "Wireless traffic modeling and prediction using seasonal arima models," *IEICE Transactions on Communications*, vol. 88, no. 10, pp. 3992–3999, 2005.

[50] W.-C. Hong, "Application of seasonal svr with chaotic immune algorithm in traffic flow forecasting," *Neural Computing and Applications*, vol. 21, pp. 583–593, 2012.

[51] B. Zhou, D. He, and Z. Sun, "Traffic modeling and prediction using arima/garch model," in *Modeling and Simulation Tools for Emerging Telecommunication Networks: Needs, Trends, Challenges and Solutions*, (Boston, MA, USA), pp. 101–121, Springer, 2006.

[52] J. Feng, X. Chen, R. Gao, M. Zeng, and Y. Li, "Deeptp: An end-to-end neural network for mobile cellular traffic prediction," *IEEE Network*, vol. 32, no. 6, pp. 108–115, 2018.

[53] P. Charles, "IMDEA-OWL." `https://github.com/cn0xroot/LTE`, 2023.

[54] IMDEA, "The IMDEA Software Institute." https://software.imdea.org/.

[55] S. Kariyappa, H. Tsai, K. Spoon, S. Ambrogio, P. Narayanan, C. Mackin, A. Chen, M. Qureshi, and G. W. Burr, "Noise-resilient dnn: tolerating noise in pcm-based ai accelerators via noise-aware training," *IEEE Transactions on Electron Devices*, vol. 68, no. 9, pp. 4356–4362, 2021.

[56] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature Communications*, vol. 11, no. 1, p. 2473, 2020.

[57] D. J. Mountain, M. R. McLean, and C. D. Krieger, "Memristor crossbar tiles in a flexible, general purpose neural processor," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 137–145, 2017.

[58] J. Shao and J. Zhang, "Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems," in *IEEE International Conference on Communications Workshops (ICC Workshops)*, (Dublin, Ireland), pp. 1–6, IEEE, June 2020.

[59] M. Jankowski, D. Gündüz, and K. Mikolajczyk, "Wireless image retrieval at the edge," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 89–100, 2020.

[60] IBM Research, "PCM statistical model — IBM analog hardware acceleration kit." `https://aihwkit.readthedocs.io/en/v0.9.0/pcm_inference.html`.

[61] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[62] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, (Santiago, Chile), pp. 945–953, Dec. 2015.

[63] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[64] N. Doan, S. A. Hashemi, and W. J. Gross, "Neural successive cancellation decoding of polar codes," in *IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, (Kalamata, Greece), pp. 1–5, IEEE, June 2018.

[65] S. Cammerer, T. Gruber, J. Hoydis, and S. Ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *IEEE Global Communications Conference (GLOBECOM)*, (Singapore), pp. 1–6, IEEE, Dec. 2017.