UNIVERSITY OF CALIFORNIA

Los Angeles

Next-generation AI: From Algorithm to Device Perspectives

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Electrical and Computer Engineering

by

Albert Lee

2021

ABSTRACT OF THE DISSERTATION

Next-generation AI: From Algorithm to Device Perspectives

by

Albert Lee
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2021
Professor Kang L. Wang, Chair

In recent years, neural networks have contributed significantly to the advancement of machine learning, achieving state-of-the-art over a broad range of challenging tasks. The world right now is seeing a global artificial intelligence (AI) revolution involving academic and industry alike: tech giants like Google and Microsoft are applying machine learning in their commercial products, while professors from every discipline- computer science, engineering, mathematics, biology, transportation - scramble to apply these methods to advance their research. Stock analysts are using AI to analyze and predict stock prices, medical experts to diagnose and develop new drugs, while game developers create sophisticated, human-like behavior in characters. At the national level, all sponsored research agencies, both NSF and DARPA have identified AI as one of the major national research directions.

Our research targets the advancement of next-generation AI from three vertical aspects along the computing hierarchy: At the algorithm level, we propose the use of application-specific, bio-inspired neural networks for information processing. We develop models of special signal-processing neurons that are compatible with today's machine learning algorithms; and optimize neural architectures containing these neurons to understand their role in creating an efficient network. At the hardware level, we address the memory bottleneck in AI accelerators. We propose two schemes to overcome limitations caused by variations in critical paths and fabrication processes. At the single device level, we recognize the significant performance gain from devices that compose AI computation via physical mechanisms. We propose two spintronic structures capable of computing convolutions that achieve orders of magnitude higher efficiency than state-of-the-art technology. These innovations provide the foundation for higher performance and more efficient AI at different time scales throughout the coming decades: in the short term, algorithms that can be implemented immediately; in the mid-term, hardware designs that can be realized in a few years; and in the long term, new device technologies to be adopted as the fabrics of AI computation.

The dissertation of Albert Lee is approved.

Yingnian Wu

Ken Yang

Puneet Gupta

Kang Lung Wang, Committee Chair

University of California, Los Angeles

2021

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENT

First of all, I would like to thank my advisor Dr. Kang Wang, an inspiring lecturer and scientist. He is my example of life-long learning, and I sincerely look up to how he continues to be extremely devoted to science despite his busy schedule. Academically, Prof. Wang allows his students to freely explore while putting us in touch with a large variety of ideas. I learned a lot from ideas that worked out, and perhaps even more from those that didn't. In addition, Prof. Wang's grand visions taught us how to view things from a greater perspective and consider about how technology interacts with the world. Throughout my Ph.D. degree, I have learned skills towards every aspect of my career path- critical thinking towards research, managing relationships with collaborators, interacting with the industry, funding for projects, intellectual properties for startups, and many more. Prof. Wang is truly the one who shaped me into the researcher I am today. On the same note, I'd like to thank Mrs. Wang, who I had the privilege to work with and learn from on various occasions. I'm a big fan of her turkey recipe as well.

I would also like to thank my committee members, Prof. Yang, Prof. Gupta, and Prof. Wu, for taking the time to discuss and go through my research and provide me with valuable advice. I would further like to thank Prof. Gupta and Prof. Pamarti, with whom I had the chance to work with very closely; and Mrs. Katie Christensen, who has been fully supportive of the group, working tirelessly with us on tight deadlines. Finally, I wish to thank Prof. Schwartz, a friendly and knowledgeable professor who serves as a secondary mentor to our subgroup, and Prof. Khalili, who introduced me to the group's magnetic technologies.

It has been an incredible experience working with the top minds at DRL. I would like to thank, in chronological order: Dr. Hochul Lee, a talented and devoted researcher. We've had countless inspiring discussions that lead to a major part of my publications, patents, and proposals. I also had the chance to participate in many of his life events (including his wedding), and I consider him to be not only a role model but also a true friend. Likewise, I thoroughly enjoyed working with Mr. Farbod Ebrahimi, a brilliant engineer and a serial brainstormer who actively pursues higher expectations for himself. He is one of my best friends in LA and an amazing person to have deep conversations with regardless of your perspective. I have worked on countless occasions with Dr. Di Wu, who is a devoted scientist and an exceptional negotiator. He is continuously enthusiastic about bringing technologies to real-life applications, working around the clock available nearly 24-7. Mr. Bingqian Dai, the group mascot with great theoretical knowledge and is very supportive of new ideas. Di and Bingqian are my major collaborators towards the end of my Ph.D. degree. While it is impossible to list everyone that has inspired me during my degree, some of the others whom I had the chance to interact with include: Dr. Wenyuan Li, Dr. Xiang Li, Dr. Rick Tsai, Dr. Bald Tsai, just recently a doctor Yuching Hsiao, not-a-doctor-yet Ping-Keng Lu, Jiyue Yang, Dr. Chin-Chung Chen, etc…. I wish all a successful carrier.

Finally, I would like to thank my family and those who I consider to be. You have been by my side through tough times and hardship; when I have been stubborn, immature, selfish, and bad-tempered. Thank you for being by my side through the years, and for being generous when I refused to admit wrong. I am not perfect, but I truly care for you and wish could do/have done more. I would specifically like to thank Ximeng Fan, who has taught me to be optimistic and to bring joy to those around me. We share many interests and viewpoints, and we can be civil on the ones we disagree with. Thank you for dealing with some of my stupid life decisions, both prior and moving forward; and for supporting me throughout.

VITA

Education

2013-2015 M.S. in Electrical Engineering
National Tsing-Hua University
Hsinchu, Taiwan

2009-2013 B.S. in Electrical Engineering and Computer Science
National Tsing-Hua University
Hsinchu, Taiwan

Employment history

2016-2021 Circuit Design Engineer
Inston Inc.
Los Angeles, California

2015-2021 Graduate Student Researcher
Department of Electrical and Computer Engineering
University of California, Los Angeles
Los Angeles, California

2017-2017 Teaching Assistant: Logical Design of Digital Systems
Department of Electrical and Computer Engineering
University of California, Los Angeles
Los Angeles, California

2014-2015 Internship: Circuit Designer
United Semiconductor Corporation
Flash Memory Circuit Design Team
Hsinchu, Taiwan

2014-2015 Teaching Assistant: An Introduction to VLSI; Embedded Memory Design
Department of Electrical Engineering
National Tsing-Hua University
Hsinchu, Taiwan

2013-2015 Graduate Student Researcher
Department of Electrical Engineering
National Tsing-Hua University
Hsinchu, Taiwan

2010-2010 Internship: Circuit Designer
Novatech
Mobile Design Team
Hsinchu, Taiwan

Selected Publications

A. Lee, B. Dai, D. Wu, H. Wu, R. N. Schwartz, and K. L. Wang, "A thermodynamic core using voltage-controlled spin-orbit-torque magnetic tunnel junctions", Nanotechnology, 2021.

A. Lee, R. Jagannathan, D. Wu, and K. L. Wang, "A 2-D Calibration Scheme for Resistive Nonvolatile Memories", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 28(6), pp.1371-1377, 2020.

A. Lee, D. Wu, and K. L. Wang, "Torque Optimization for Voltage-Controlled Magnetic Tunnel Junctions as Memory and Stochastic Signal Generators", IEEE Magnetics Letters, 10, pp.1-4, 2019.

A. Lee, H. Lee, F. Ebrahimi, B. Lam, W.H. Chen, M.F. Chang, P. Khalili, and K. L. Wang, "A dual-data line read scheme for high-speed low-energy resistive nonvolatile memories", IEEE Transactions on Very Large Scale Integration Systems, 26(2), pp.272-279, 2017.

A. Lee, C. P. Lo, C. C. Lin, W. H. Chen, K. H. Hsu, Z. Wang, F. Su, Z. Yuan, Q. Wei, Y. C. King, C. J. Lin, H. Lee, P. Khalili, K. L. Wang, Y. Wang, H. Yang, Y. Liu, and M. F. Chang, "A ReRAM-Based Nonvolatile Flip-Flop With Self-Write-Termination Scheme for Frequent-OFF Fast-Wake-Up Nonvolatile Processors", IEEE Journal of Solid-State Circuits, 52(8), pp.2194-2207, 2017.

A. Lee and K. L. Wang, "Full Memory Encryption with Magnetoelectric In-Memory Computing", In 2019 International Symposium on VLSI Technology, Systems and Applications, pp.1-2, 2019.

A. Lee, C. C. Lin, T. C. Yang, and M. F. Chang, "An embedded ReRAM using a small-offset sense amplifier for low-voltage operations", In 2015 VLSI Design, Automation and Test, pp. 1-4. 2015.

A. Lee, M. F. Chang, C. C. Lin, C. F. Chen, M. S. Ho, C. C. Kuo, P. L. Tseng, S. S. Sheu, and T. K. Ku, "RRAM-based 7T1R nonvolatile SRAM with 2x reduction in store energy and 94x reduction in restore energy for frequent-off instant-on applications", In Symposium on VLSI Circuits, pp. C76-C77, 2015.

Accepted Patents

[US10861527B2], [US10255976B1], [US9972400B1], [US9722584B1], [US9502114B1], [US9564209B1]

CHAPTER 1

INTRODUCTION

In this research, we address the advancement of next-generation AI from three vertical perspectives: At the algorithm level, the development of bio-inspired neural networks with neurons specialized for information processing; at the hardware level, improved memory design for AI accelerators; and at the single device level, spintronic devices that compose AI computation via physical mechanisms. This dissertation is organized as follows:

Chapter 1 gives an overview of the three perspectives. In Sec.1.1, We go over the historical development of AI where there were periods of booming interest and as well as times when there was lack of resources in the academia, industry, as well as the government. We review the historical reasons and events that lead to widespread AI enthusiasm, the challenges that lead to its temporary downfalls, and how prior developments affected AI as it is today. In Sec.1.2, We discuss the hardware accelerators for efficient, high-performance AI computation, from CPUs and GPUs to ASICs and memristive crossbars. In Sec.1.3, we review the history of spin-based mechanisms, recent discoveries, and their applications.

Chapter 2 presents our algorithmic approach of adopting bio-inspired signal-processing neurons in machine learning. In Section 2.1, we discuss our motivation and show examples of neuronal behavior that are critical to information processing in the human brain. In Sec.2.2, we present a modelling algorithm that is used to create models of the the signal-processing neurons. To understand the role of these neurons, we propose three bio-inspired neural architecture search (NAS) algorithms in Sec.2.3, e.g. genetic mutation, neural growth, and neural pruning. These NAS algorithms allow us to study optimized network architectures containing the signal-processing neurons as the amount of neurons in the network stays constant, increases, and decreases. Sec.2.4 describes the setup for our experiments, which include data perparation, the parameters for the NAS, and the parameters for training. Sec.2.5 presents analysis results on the optimized network architectures, revealing general structures and the organization of the neurons. These results are used to create a new efficient and high-performance network.

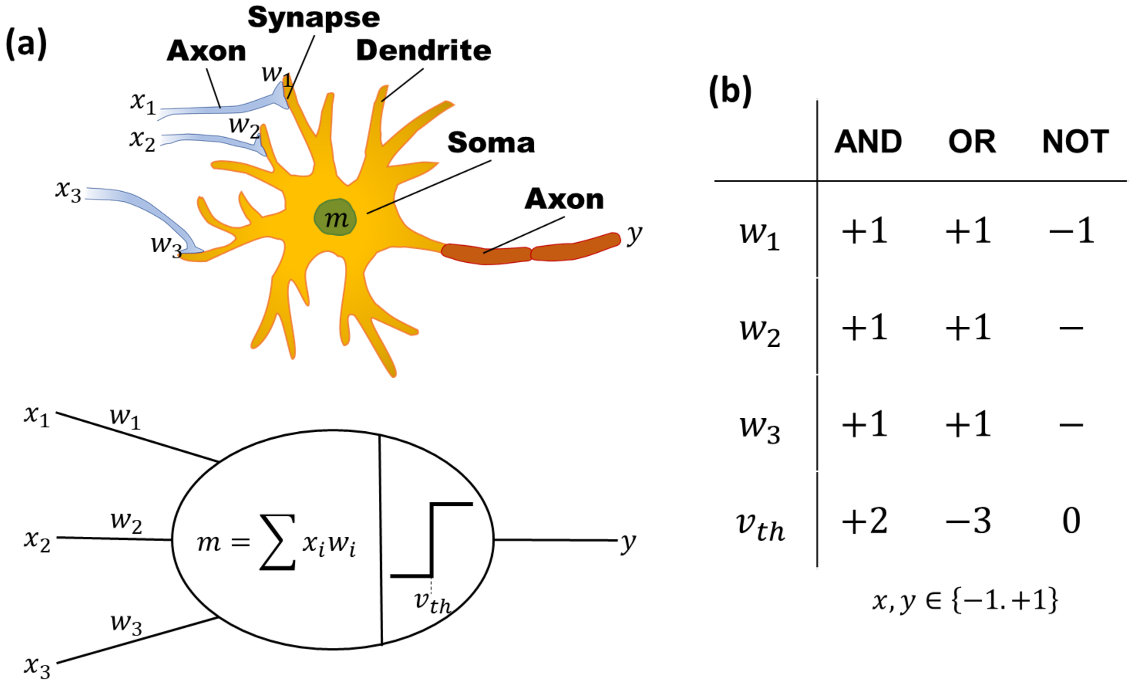
Chapter 3 presents our approach for improving memory performance in AI accelerators. Section 3.1 discusses the importance of memory in AI accelerators and why they are the bottleneck in computation performance. In Section 3.2, we give an introduction on memory architecture, design, and challenges, with focus on sources of variability that limit memory performance. We then propose two approaches to overcome variation. Section 3.3 describes the 2D calibration scheme, in which an improved calibration grid compensate for variations across an array. Section 3.4 describes the dual dataline sensing scheme, in which the circuit's tolerance of variability is enhanced though a higher sensing margin. In both schemes, we show the high-level desrciption, circuit implementation, and the simulated performance.

In chapter 4, we present our spintronic device capable of high-efficiency signal processing. Sec.4.1 outlines our motivation, which discusses the advantages of spintronics over electronics such as the flexibility in computing mechanism and nonvolatility. Sec.4.2 give a background on the problem that our device aims to solve (e.g. convolution) and the spin mechanisms that are used. In Section 4.3 and 4.4, we present two implementations of the spintronic signal processor, each based on a combination of various spin mechanisms. In both designs, we show the device structure, the theoretical performance, and the experiment or simulation results.

Finally, we wrap up this dissertation with a conclusion and a discussion of future works in the direction of the presented research in Chapter 5. We also present our perspectives towards the future of AI.

1.1 History of Neural Networks

The large number of recent breakthroughs make artificial neural networks seem like a recent discovery, yet its development actually dates far back with its fair share of ups and downs. The earliest artificial neurons can be traced back to 1943 when Walter Pitts and Warren McCulloch, a neuroscientist and a logician, teamed up to create a mathematical model of a neuron in the human nervous system [1]. This model, shown in **Fig.1.1.1a**, captured the operation of the neuron in two parts: (1) The integration of input signals coming from axons, weighted by the strength of synapses as the potential of the neuron body (Soma); (2) The response of the neuron potential, goingwhich goes through a rapid depolarization when it reaches a threshold. The McCulloch-Pitts neuron model can thus be described as a process with inputs x_i, each weighted by their respective weights w_i, integrated as the neuron potential $m = \sum x_i w_i$; then passing the potential through a binary threshold function as the neuron's output (e.g. $y = 1\ if\ m > v_{th}\ else -1$). The authors further showed that the model is capable of modelling universal



	AND	OR	NOT
w_1	$+1$	$+1$	-1
w_2	$+1$	$+1$	–
w_3	$+1$	$+1$	–
v_{th}	$+2$	-3	0

$x, y \in \{-1.+1\}$

Fig.1.1.1 Biological neuron and the McCulloch-Pitts model. (a) In a neuron, the potential of the soma is the sum of the signals from its input axons weighted by synapses. When this potential exceeds a threshold, a pulse is sent to the output axon. This procedure is modelled in the McCulloch-Pitts neuron via integration of weighted inputs and a binary threshold function. (b) Weight and threshold design to implement the AND, OR, and NOT logical functions with a McCulloch-Pitts neuron.

logic functions AND, OR, and NOT by designing the neurons weights and threshold, as shown in **Fig.1.1.1b**. For example, consider binary imputs and outpus of ± 1. A 3-input neuron with weights of all $+1$ and a firing threshold of -3 would output $+1$ if any input is $+1$, and only -1 if all its inputs are -1. This neuron thus implements a 3-input OR function.

Later, Frank Rosenblatt combined the McCulloch-Pitts neuron with biological learning rules discovered to develop the perceptron [2]. By applying the learning rule discovered by neuroscientist Donald Hebb: “Neurons that fire together, wire together” [3], the perceptron promised learning capabilities of achieving binary classification on its own. Rosenblatt optimistically predicted that the perceptron “may eventually be able to learn, make decisions, and translate languages”. Soon afterward, Alexey Grigoryevich Ivakhnenko and Valentin Grigorʹevich Lapa proposed a hierarchical representation now considered the first multi-layer network [4].

Optimism in the bright future of AI led to a surge in funding, research interest, and, unfortunately, inflated promises on its abilities. When many of these claims failed to materialize, concerns soon arose across both funding agencies and the academia. The US government, initially interested in its ability to automatically translate Russian documents, became concerned about the lack of progress in automatic machine translation despite millions of investments. In 1964, the national research council (NRC) formed the automatic language processing advisory committee (ALPAC) to look into this issue; and concluded that machine translation was more expensive, less accurate, and slower than human translation [5]. Similarly, in the UK, Professor James Lighthill, tasked by the British Science Research Council to evaluate the state of AI research, concluded that “In no part of the field have the discoveries made so far produced the major impact that was then promised” [6]. These events led to a wide termination of funding support for AI. Academically, Marvin Minsky and Seymour Papert outlined the theoretical limitations of perceptrons in their book “Perceptrons”, resulting in a sharp drop in AI interest [7]. The lack of funding and academic interest led to a period now known as the first AI winter [8].

Research in AI revived in the mid-80s with Paul Werbos's backpropagation algorithm [9], which enabled practical, efficient training of multi-layered networks by computing error gradients through the chain rule and modifying the weights at each node accordingly. This renewed interest led to new discoveries that compose the core of many of today's AI algorithms. In 1979, Kunihiko Fukushima developed the Neocognitron based on the structure of the human primary visual cortex [10]. It demonstrated the ability to recognize visual patterns as the first convolutional neural network architecture. The Hopfield Network [11] developed by John Hopfield in 1982, made recurrent structures popular and showed promise as dynamic content-addressable memory. Q-learning presented by Christopher Watkins [12] laid the groundwork for reinforcement learning via the introduction of states, actions, and delayed rewards.

These new developments brought about the commercialization of a form of AI called the *Expert system*. The *Expert system* aims to emulate the decision-making process of a human expert to solve complex problems and make sequential decisions by reasoning through a knowledge database. In the early-mid 1980s, *Expert systems* such as XCON showed enormous commercial success and were adopted by companies around the world. Billions of dollars poured into the industry, supporting software companies like Techknoeldge, Intellicorp, and hardware companies such as Symbolics and LISP (List-Processing) machines.

However, *Expert systems* proved to be too expensive to maintain and update. They were rigid and fragile, and competition from general purpose systems soon brought down their market. Workstations from Sun Microsystems provided alternatives just as powerful but much more flexible, while personal computers from Apple and IBM provided a simpler and (through development) architecture that achieved similar results. With the market share of *Expert* *systems* taken *away*, their demand collapsed as quickly as it rose. By the early 1990s, nearly all commercial *Expert system* companies had failed.

This commercial failure resulted in a second blow to the development of AI. In this second AI winter, many researchers deliberately avoided the term AI, using alternate terms like informatics, intelligent systems, and knowledge-based systems. Some believed that their works were fundamentally different from what was previously called AI, while some simply didn’t want the reputation that came with the name. Nevertheless, the data-driven approach of Expert systems significantly influenced AI algorithms built today.

The recent revival of AI in the 2000s has much to do with the exponential increase in computing power from the scaling of silicon chips. Initially, brute-force approaches demonstrated superhuman performance across various tasks. Deep Blue, with the ability to compute 200 Million moves per second, became the first computer system to beat the world chess champion in 1997 [13]. Data became a driving force of AI with Yann LeCunn’s release of the MNIST handwritten digits database in 1998 [14]. MNIST provided a benchmark for evaluating the performance of AI algorithms and remains widely used even today. The Imagenet database developed by Fei Fei Li [15] and its associated ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is considered by many the catalyst for the AI boom. Since 2012, neural networks dominated the challenge with the Alexnet victory, significantly outperforming other competitors by over 10 percentage points [16]. Through data, AI systems began solving challenges that brute force could not: in 2016, Google's AlphaGo, a neural network trained using reinforcement learning, becomes the first program to beat an unhandicapped professional human player in Go [17]. This feat was previously thought impossible for another 30 years as a brute force approach would have to evaluate more possibilities than the number of atoms in the universe (10^{82}).

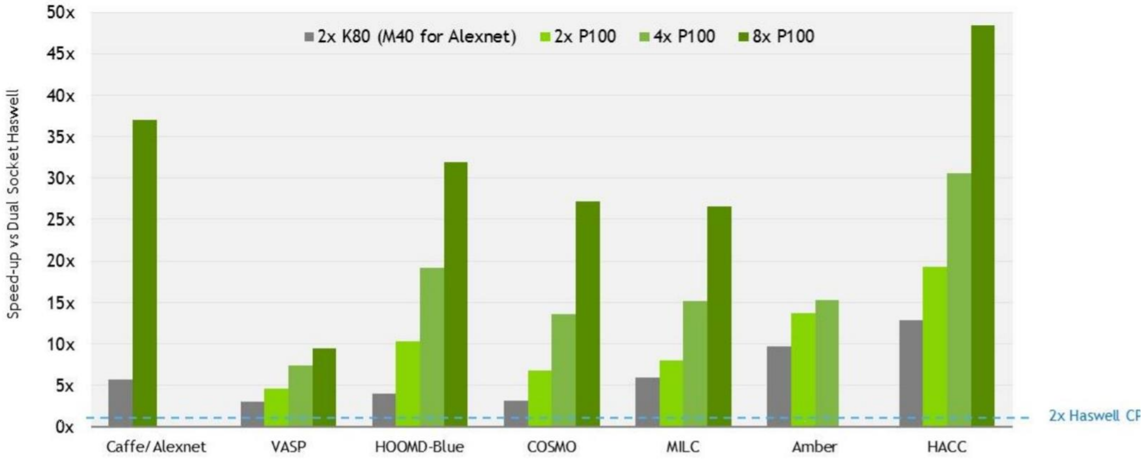


Fig.1.2.1 Speedup of various AI platforms on a GPU. Figure adapted from Nvidia's Tesla P100 White Paper [17].

Today, our progress has far exceeded the wildest claims that brought about the 1st AI winter. Discussions surrounding the abilities of AI have shifted from questioning whether it will be useful to whether we restrict its abilities. Nevertheless, as we look towards the future, AI aims to be involved in everyday life; from tasks as small as recommending new songs to complex medical problems that trouble even top experts with decades of experience.

1.2 Neural Network Hardware Accelerators

The increasing demand for AI has led to the development of specialized accelerator hardware. Traditional central processing units (CPU) that were optimized for flexibility rather than parallelization did not perform well on AI tasks. On the other hand, graphic processing units (GPUs) have been able to massively parallelize AI tasks as they were built to process similar tensor (multi-dimensional vector) operations. Empirical studies show that GPUs provide over 30x speedup over state-of-the-art CPUs **(Fig.1.2.1)**, making them the top choice for AI tasks [18]**.** In addition, GPU producers have also committed to developing specialized toolkits and modifying tensor cores to further accelerate AI computation [19]. These packages have been adopted by the majority of mainstream AI design platforms [20] [21].

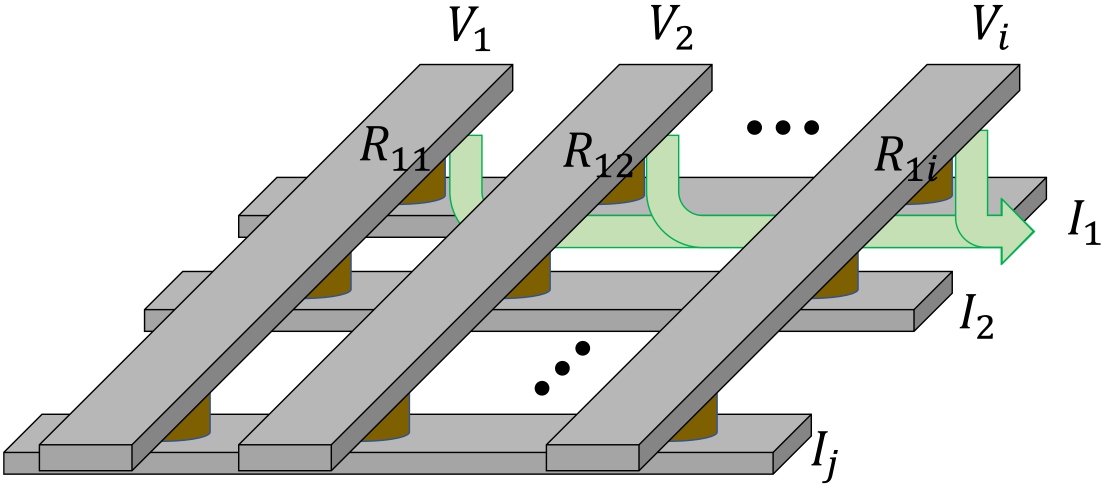


Fig.1.2.2 Physical implementation of matrix multiplication via a crossbar of memory devices. Voltages $V_1 .. V_i$ are placed on vertical wires connected to the top electrode of memristors with resistances R_{11} ... R_{1i}. From the Ohm's law, each device has a current of V_k/R_{1k} . The bottom horizontal wire sums the current flowing through the memristors upon it, e.g., $I_1 = \sum_i V_k/R_{1k}$. The full array entire array can be represented in matrix form as $I = R^{-1}V = GV$.

Compared to GPUs, application-specific integrated circuits (ASICs) can further boost the efficiency of AI computation. These accelerators discard generalized capabilities of GPUs and optimize their data flow, data format, and memory bandwidth purely for AI-related tensor processing. Many data centers today have adopted this approach with chips such as the Google TPU [22] and Baidu Kunlun processor [23] being developed; while Amazon and Facebook reportedly in the process of designing their own ASICs. Taking this approach further, designs for reduced bit counts [24]–[27], convolutional networks [28], [29], stochastic networks [30], generative adversarial networks [31], and recurrent networks [32] have also been developed.

With the recent progress in new memory technologies, researchers have also targeted the physical implementation of matrix multiplication using memory devices [33]–[37]. This approach, shown in **Fig.1.2.2**, composes a crossbar array of memory units with wires running vertically on top of the array and horizontally beneath. When voltages V_1 ... V_i are placed on the top wires, the currents I_1 ... I_j flowing through the bottom wires can be computed via Ohm's law and Kirchhoff's current law, as the sum of V_k times the conductance $G_{jk} = 1/R_{jk}$ of each memory unit, e.g. $I_j = \sum_i V_k G_{jk}$. Considering the entire array, the bottom wire currents can be expressed as in matrix form as $I = GV$. This architecture allows efficient matrix multiplication in a single cycle and significantly reduces data access across memory hierarchies.

In the advancement of AI hardware, the effect of the memory bottleneck on computing performance has become increasingly evident. As much as designs attempt to avoid the memory bottleneck (also known as the Von-Neumann Bottleneck), powerful algorithms and networks continue to demand more memory. Today's networks are often orders of magnitude larger than those used a decade ago [38]. Hundreds of megabytes are needed for weight storage alone, and an order more for computing gradients and intermediate values during training. As a result, addressing memory limitations is a critical issue in further advancing AI.

1.3 Spintronic Technology

Spintronics is an emerging research field that relates to the study of the electron spin and its associated magnetic moment. The theoretical foundation of spintronics was developed by Wolfgang Pauli via Pauli matrices and quantum mechanics in the 1920s [39], which also described its relation to the quantized electron spin. Subsequently, a variety of spin-related theories have been proposed, including that of the Landau–Lifshitz–Gilbert dynamics [40] in 1935 and the prediction of the spin Hall effect in 1971 [41].

Spintronic applications took a huge step forward with the discovery of magnetoresistance effects in the 80s from tunneling experiments. Meservey and Tedrow's experiments on a ferromagnet/superconductor stack [42] and Jullière's experiments on Fe/Ge/Co stack [43] discovered the conservation of spin and the tunneling magnetoresistance effect (TMR). With the development of thin-film fabrication tools, Peter Grünberg and Albert Fert showed the giant magnetoresistance (GMR) in 1988 [44]. Since then, there has been an explosion in the discovery of spintronic mechanisms, including the spin-transfer torque effect (STT) [45], [46], Spin-Orbit Coupling (SOC) effects (voltage-controlled magnetic anisotropy effect (VCMA) [47]–[49] and spin-orbit torque (SOT) [50]–[52]); spin textures (skyrmions [53]–[55], spin fluid [56]–[58], spin glass [59]–[61], and magnetic domains[62]–[64]); spin dynamics (Dzyaloshinskii–Moriya interaction [65]–[67], spin waves [68]–[70]); spin-thermal effects (Spin Seebeck [71]–[73]); and spin-optical effects (Magneto-optical Kerr [74]–[76]).

Today, spintronics is widely used in data storage and manipulation: spins provide an efficient, compact way of storing nonvolatile data in its orientation; and GMR/TMR-based sensors read out the magnetization as electrical signals. This technology brought about the development of hard drives and a drastic increase in the data storage density. In the 2000s, high tunneling magnetoresistance (TMR) magnetic tunneling junctions (MTJs) [77]–[79] have also received great interest as an integrable, nonvolatile random access memory element with high speed and endurance. Switching mechanisms have evolved from using magnetic fields to spin-based mechanisms (**Fig.1.3.1**), with STT-MRAM currently being developed by major foundries worldwide [80]–[82], and VCMA [83]–[85], SOT [86], [87] as the next-generation memory. Current research on MTJs focuses on two aspects: (1) high TMR structures with improved readability through enhanced interface characteristics and stronger polarization; and (2) energy efficient switching through enhancement of spin mechanisms.

The abundance of spin mechanisms has also led to proposals for unconventional, nonbinary computing schemes (**Fig.1.3.2)**. For example, coupling between magnetic elements through dipolar interaction, the Dzyaloshinskii-Moriya interaction, as well as spin waves has been proposed for optimization by utilizing the energy minimization process of the interacting bits [88]–[93]. Nonlinear interactions between spintronic elements have been used for complex mapping reservoirs built upon skyrmions, MTJs, and oscillators [94]–[98]; for composing filter-like functions using spin-wave sources, lenses, and mirrors [99], [100]. Neuromorphic computing such as physical crossbars and nonlinear activations have been composed using

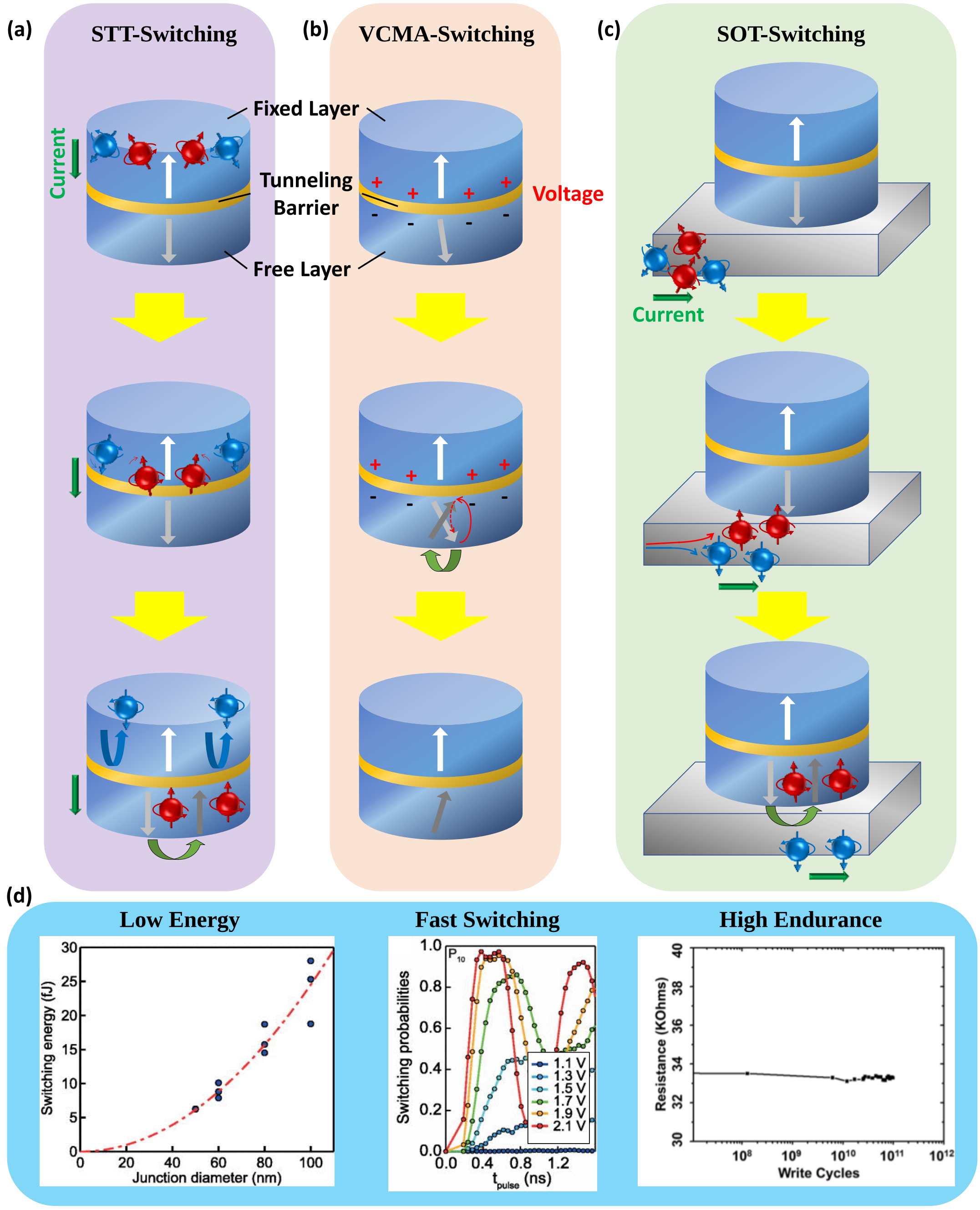


Fig.1.3.1 MRAM operation mechanisms: (a) STT Switching. Electrons passing through the free layer transfer their spin angular momentum to the moments of the magnetic free layer, thereby switching its state. (b) VCMA switching. A voltage placed across the MTJ modifies the anisotropy of the magnetic free layer, leading to a precessional (damped oscillation) motion of the free layer magnetization. (c) SOT switching. The MTJ is placed upon a bus with high spin-orbit coupling. As current passes through the bus, electrons with different spin orientations accumulate on the sides of the material, which then transfer their spin moment to the free layer to change its state. (d) Advantages of magnetic memory: scalable, low energy, sub-ns switching, and high endurance with no degradation over 10^{11} cycles. Subfigures adapted from [83].

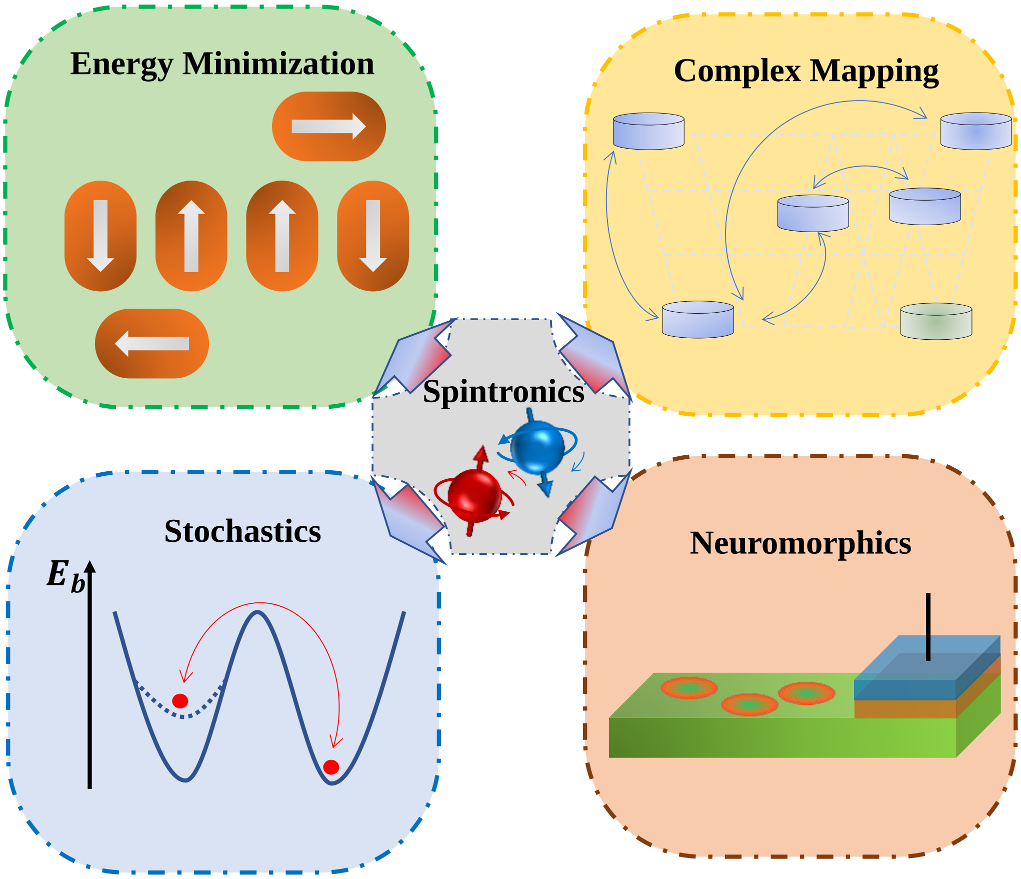


Fig.1.3.2 Illustration of spin-based nonvonventional computing methods. Energy minimization schemes compose a system of coupled spin elements so that the lowest energy state represents the solution. Complex mapping uses spin elements to map inputs to different signals, such as reservoirs or high-dimensional computing. Neuromorphic devices such as synapses have been built upon spin elements, and likewise, Spin-based stochastics have utilized fluctuations in device states for computing.

domains and MTJs [101]–[105]; skyrmions have been proposed to model neurotransmitters [106]; finally, spin-stochastics have been used in probabilistic computing [107], [108], thermodynamic computing [109], and many more. However, the majority of these methods were unable to provide a definite advantage over existing CMOS implementations, and many were difficult to realize at a scaled industrial level. Our research aims to address this gap.

CHAPTER 2

SIGNAL-PROCESSING NEURAL NETWORKS

In this chapter, we present our work on utilizing special signal processing neurons to improve neural network performance. The chapter is outlined as follows: we first review different neuronal behavior and selected signal processing neurons to model. We then use neural architecture search (NAS) to optimize network architectures containing the neurons, and analyze the optimized architectures to extract characteristics of the networks. These results are then used to develop the signal-processing neural network.

Section 2.1 describes our motivation and outlines how neurons with specialized functions play a crucial role in information processing within the human brain. Chapter 2.2 reviews the functions two neurons: that of the *max* neuron in the visual cortex (Sec.2.2a) and the *coincidence* neuron in the audio cortex (Sec.2.2b), along with how they associate with neural network design today. Then, in Sec.2.2c, we detail the step-by-step process to create the approximate piecewise-linear model of these two neurons.

With the models, we then use neural architecture search (NAS) to find high-performing architectures. In Sec. 2.3, We describe the three NAS algorithms used in this study, based upon three biological pheonmena: neural growth, neural pruining, and genetic mutation. These three NAS schemes also correspond to how the network architure changes as the amount of neurons in the network stays constant, increases, or decreases. Sec.2.4 then details the setup of the experiments, e.g., the datasets we use, how they are preprocessed, and parameters of the search and training process. Analysis results of the optimized architectures is presneted in Sec.2.5, in which we illustrate how the network structure, amount of connections, and types of neurons change throughout the network. The analysis results are then used to design a compact, high-performance signal processing neural network.

2.1 Motivation

As described in Chapter 1, advancements in artificial neural networks have often taken inspiration from biology. For example, the McCulloch-Pitts neuron arose from modeling of the neuron’s integrate and fire behavior; convolutional network structures from the visual cortex; and network compression methods from neural pruning.

In the past decade, a large variety of neuronal behavior has been discovered. Neurons have been shown to differ in signal transmission (bipolar, unipolar, and multi-polar), functionality (sensory, motor, and interneuron), structures (connection shapes and density), as well as the type of neurotransmitter and neuromodulators involved [110]. Specialized neurons form crucial functions in information processing systems that are shared throughout organisms rather than generated through learning. For example, grid cells in the entorhinal cortex play a major role in composing the brain’s cognitive spatial map [111]–[114]. They operate through the phase precession mechanism, in which the firing of a neuron is timed in relation to the phase of the neural oscillation in the surrounding cells [115]. At the behavioral level, these neurons fire when their host occupies a spatial grid in the environment (**Fig.2.2.1**). Grid cells have been

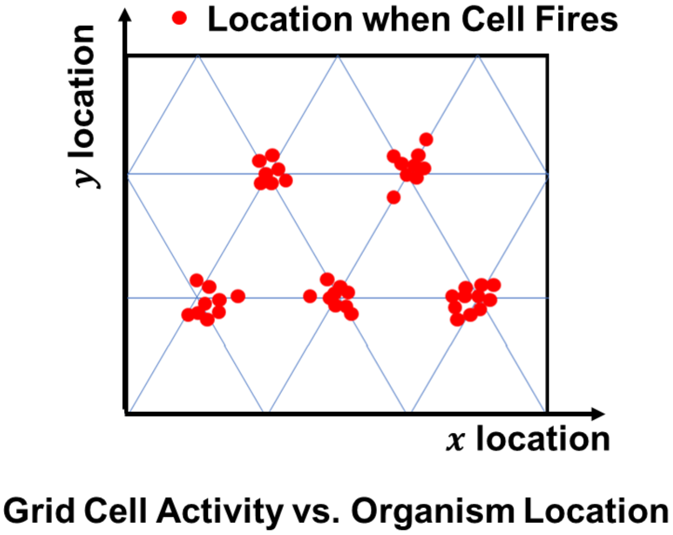

● Location when Cell Fires
y location
x location
Grid Cell Activity vs. Organism Location

Fig.2.1.1 Firing activity of grid cells. Grid cells fire regularly when its host organism occupies points on a spatial grid. Their firing activity contribute towards the spatial cognitive map in the brain by providing information about location and distance.

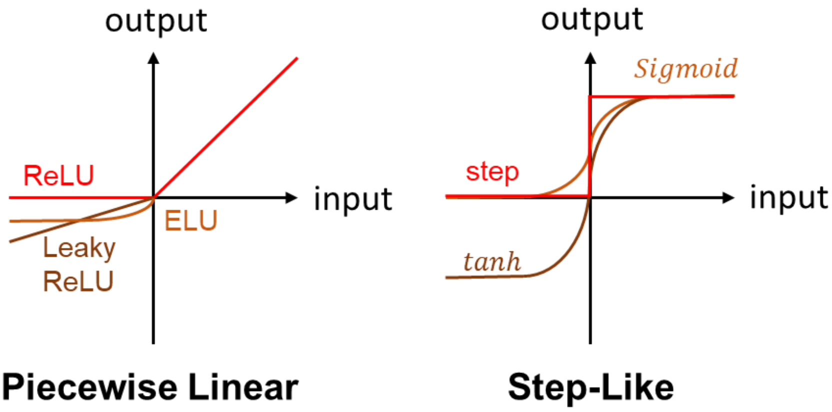


Fig.2.1.2 Common activation functions used in today's neural networks. (Left) the family of piecewise linear units: Rectified Linear Units (ReLU) rectifying negative signals to zero, leaky ReLU has a nonzero slope in the negative region, and Exponential Linear Units (ELU) has an exponential slope in the negative region. (Right) the family of step-like activation functions take forms of the binary step, $sigmoid$, and $tanh$.

discovered across a variety of higher cognitive mammals including mice, monkeys, and humans.

Exploring and utilizing neuronal functions provide a promising path to boost machine learning performance, and could eventually lead to an improved understanding of the biological information processing flow. However, today's artificial neural networks (ANNs) have yet to take inspiration from these discoveries. Neuronal responses, often characterized by a neuron model's activation function, have settled to a few nonlinear functions after a long process of numerical and empirical exploration (**Fig.2.2.2**). From the binary-thresholding function used in the McCulloch-Pitts neuron, activation functions first shifted to continuous functions like sigmoid and tanh that enable backpropagation. Piecewise linear units later took the center stage due to their non-saturating gradient and low computation complexity. Rectified Linear Units (ReLU) celebrated success in image recognition [116]–[118], while those with a nonzero negative slope (Parameterized ReLU (PReLU), Exponential ReLU (ELU), LeakyReLU [119], [120]) have been used in image generation. c

2.2 Neuron Models

2.2a Max Neurons

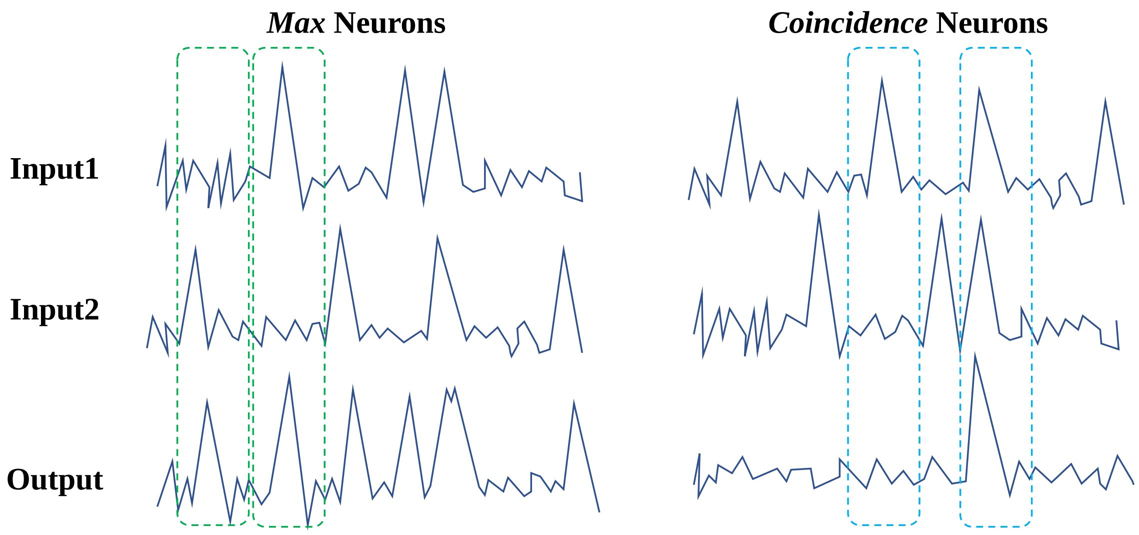


Fig.2.2.1 The distinct features and behavior of *max* and *coincidence* neurons. A *max* neuron outputs the maximum value among its inputs, and fires whenever any of its inputs fire (green box). A *coincidence* neuron detects coincidence in its inputs, and fires only when both inputs are simultaneously active (blue box)

Signal processing in the visual cortex is mainly composed of a hierarchical buildup of receptive fields, in which larger receptive fields of latter layers can be modeled by a linear combination of the smaller receptive fields in earlier layers. However, in some of the higher regions of the visual cortex, neuronal functions that cannot be modeled by linear combinations have also been discovered. One such function is the *max* function (**Fig.2.2.1**), in which a neuron's output is determined by the input with the highest level of activity [122], [123]. The *max* function is able to identify the dominant signal in a large number of inputs, and its commutative property results in a response that is invariant to spatial shifts within its receptive field, and reduce the impact of noise in non-dominant signals [124], [125].

Since their discovery, the adaptation of the *max* function has been abundant in image processing. Its most successful adaptation is perhaps feature pooling, in which a filter scans localized patches and outputs the maximum or average value within the region. Feature pooling is often inserted after convolutions to down-sample features and extract abstract, higher-level concepts from the applied region. They have demonstrated the same benefits as the *max* neurons, e.g., dominant signal identification, invariance to image transformations, and robustness to noise. A detailed theoretical discussion on pooling can be found in [126].

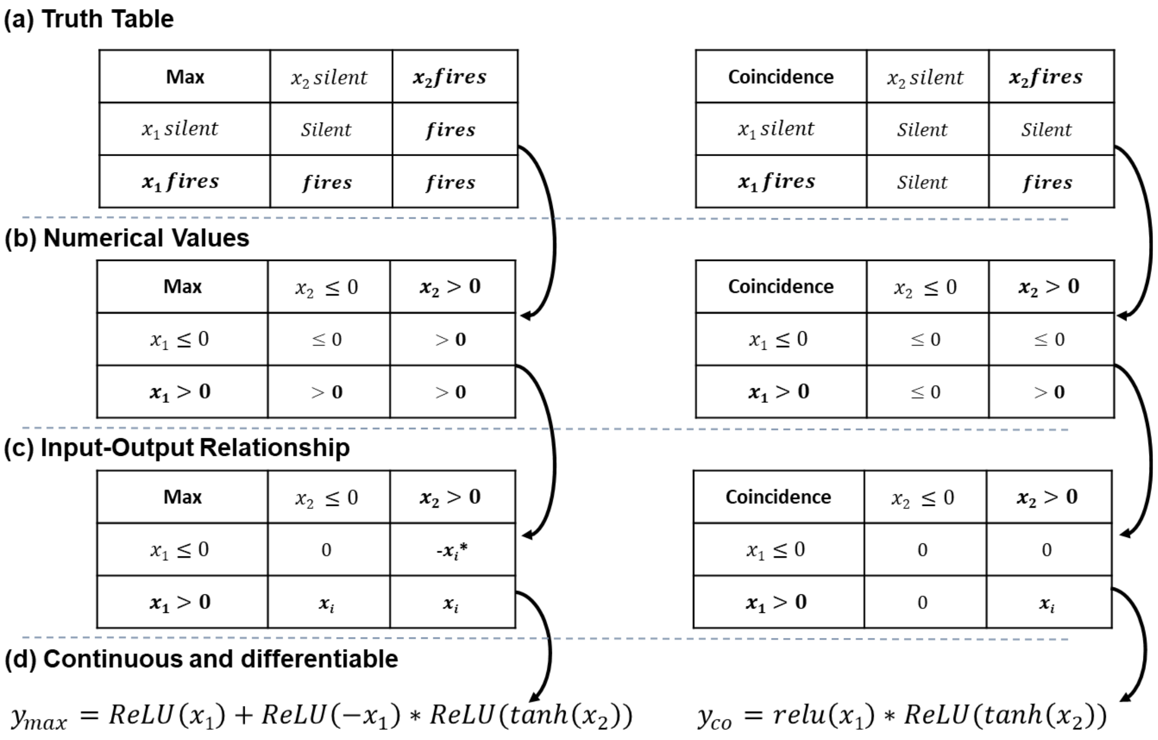
2.2b Coincidence Neurons

Studies of human auditory processing have shown that temporal and frequency structures are crucial to the representation of audio events [127], [128]. In the human auditory system, signals picked up by auditory nerve fibers excite cells that detect the coincidence, or simultaneous activity, in a group of fibers. *Coincidence* neurons have extremely low time constants and input resistance, giving them the ability to compare timing cues with accuracy over 100x of neural firing [129], [130]. They respond maximally when multiple input synapses are simultaneously active, and significantly less otherwise (**Fig.2.2.1**).

The success of gated networks and attention networks in audio processing is likely associated with the *coincidence* function. In a gated network, a neuron's output is blocked unless its gating signal is high. In other words, its output is high only when both the gating signal and its input are high at the same time. This behavior is characteristic of the *coincidence* function. Examples of these networks include Long Short-Term Memories (LSTMs) and gated Recurrent Neural Networks (RNNs) [131]–[133].

2.2c Model Algorithm

Our first step towards modeling the signal processing neurons is to represent the response of the neurons in the form of a truth table. The truth table defines the regions of linearity and boundaries where nonlinearity occurs. Here, we consider a neuron with two inputs whose activies are either *firing* or *silent*. The *max* neuron's response is the maximum value among its input synapses. In other words, its output fires whenever any of its inputs fire. On the other hand, the *coincidence* neurons only fires when all their inputs fire at the same time. The table



(a) Truth Table

Max	$x_2\, silent$	$\boldsymbol{x_2 fires}$
$x_1\, silent$	$Silent$	$\boldsymbol{fires}$
$\boldsymbol{x_1 fires}$	$\boldsymbol{fires}$	$\boldsymbol{fires}$

Coincidence	$x_2\, silent$	$\boldsymbol{x_2 fires}$
$x_1\, silent$	$Silent$	$Silent$
$\boldsymbol{x_1 fires}$	$Silent$	$\boldsymbol{fires}$

(b) Numerical Values

Max	$x_2 \le 0$	$\boldsymbol{x_2 > 0}$
$x_1 \le 0$	≤ 0	$> \boldsymbol{0}$
$\boldsymbol{x_1 > 0}$	$> \boldsymbol{0}$	$> \boldsymbol{0}$

Coincidence	$x_2 \le 0$	$\boldsymbol{x_2 > 0}$
$x_1 \le 0$	≤ 0	≤ 0
$\boldsymbol{x_1 > 0}$	≤ 0	$> \boldsymbol{0}$

(c) Input-Output Relationship

Max	$x_2 \le 0$	$\boldsymbol{x_2 > 0}$
$x_1 \le 0$	0	$\boldsymbol{-x_i^*}$
$\boldsymbol{x_1 > 0}$	$\boldsymbol{x_i}$	$\boldsymbol{x_i}$

Coincidence	$x_2 \le 0$	$\boldsymbol{x_2 > 0}$
$x_1 \le 0$	0	0
$\boldsymbol{x_1 > 0}$	0	$\boldsymbol{x_i}$

(d) Continuous and differentiable

$$y_{max} = ReLU(x_1) + ReLU(-x_1) * ReLU(tanh(x_2)) \qquad y_{co} = relu(x_1) * ReLU(tanh(x_2))$$

Fig.2.2.2 Modelling algorithm of the specialized neurons. The models process cosists of 4 steps: (a) creating the truth table of the neuron's response behavior, (b) assigning the numerical boundaries to regions based on the the truth table, (c) assigning numerical values to each region, and (d) smoothing boundaries with the $tanh$ function.

is shown in **Fig.2.2.2a**. Interestingly, the truth tables of the two neurons match that of logical functions "AND" and "OR".

After composing the truth tables, numerical ranges are assigned to define the signals that are considered as *firing* and *silent* (**Fig.2.2.2b**). Here, we follow the conventional definition of *firing* as a signal exceeding a threshold of 0 and *silent* otherwise.

$$x\ fires\ \leftrightarrow x > 0;\ x\ silent\ \leftrightarrow x \le 0$$

$$y\ fires\ \leftrightarrow y > 0;\ y\ silent\ \leftrightarrow y = 0$$

We then define the input-output relationship within each entry of the truth table. Taking inspiration from piecewise linear functions, we set the y to be linear to x_1 with a unit slope when the output is $firing$ and 0 otherwise:

$$y = |x_1| \text{ if } y\ is\ firing$$

$$y = 0 \text{ if } y\ is\ silent.$$

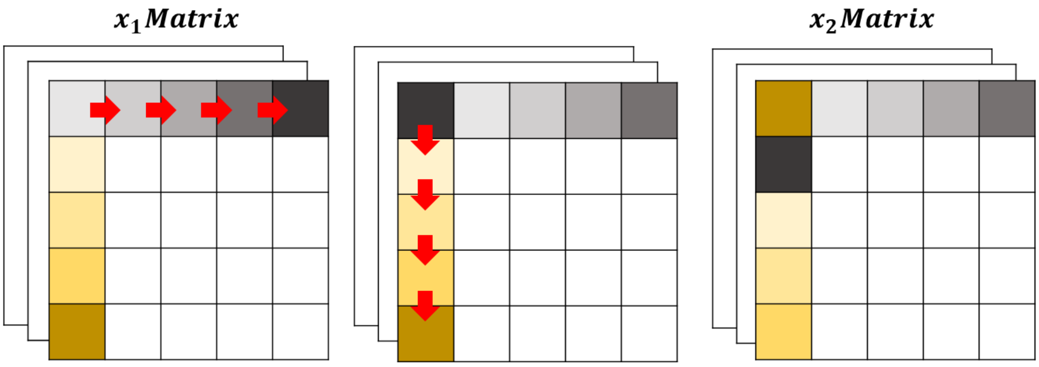


Fig.2.2.3 Obtaining a second input x_2 from a single feature map x_1. Here, we create x_2 by shifting x_1 by one element in each dimension of the feature map.

The corresponding gradients are:

$$|\partial y/\partial x_1| = 1 \text{ if } y\ is\ firing$$

$$|\partial y/\partial x_1| = 0 \text{ if } y\ is\ silent$$

Finally, as the table contains discontinuous, non-differentiable transitions, we smooth the transition between entries with $tanh$ function on x_2. With this, we arrive at the two models of the two neurons:

$$y_{co} = R(x_1) * R(T(x_2)) \quad \ldots Eq. 2.2.1$$
$$y_{max} = R(x_1) + R(-x_1) * R(T(x_2)) \quad \ldots Eq. 2.2.2$$

Where y_{co} is the *coincidence* neuron model, y_{max} is the *max* neuron model, R is the ReLU function, and T is the $tanh$ function.

The *max* and the *coincidence* neuron models have two inputs that can come from different parts of a network architecture. Where only one input is available, we can let x_2 be a manipulated or shifted version of x_1. For example, we can create x_2 by shifting x_1 by 1 element in each feature map dimension, as shown in **Fig.2.2.3**. In this case, we can reduce the functions to

$$y_{co\_s} = R(x_1) * R(T(x_1 \ll 1)) \quad \ldots Eq. 2.2.1a$$
$$y_{\max\_s} = R(x_1) + R(-x_1) * R(T(x_1 \ll 1)) \quad \ldots Eq. 2.2.2a$$

We use this reduction in our experiments in Sec.2.5.

2.3 Neural Architecture Search Algorithm

Specialized neurons are not located uniformly in neural networks. For example, *max* neurons are located in the higher regions of the visual cortex, operating upon the outputs of linear neurons in earlier visual regions. A network composed entirely of *max* neurons, whether biological or artificial, provides poor functionality as the entire network essentially reduces to a single wide *max*. Therefore, we must base our analysis on suitable network architectures. Toward this goal, we use network architecture search (NAS) to obtain high-performing architectures.

A NAS algorithm (**Fig.2.3.1**) optimizes a network by iteratively searching through an architecture space to find the best-performing architecture. It begins from a set of initial architectures; then, in each search iteration, new architectures are created and evaluated. If the new architectures perform well, they are used to update the current architectures. As a result, a high performing architecture can be obtained after multiple iterations.

A NAS algorithm can be described in three components [134]: the search space, the search method, and the evaluation strategy. The search space defines the architecture space that the algorithm searches through, and is composed of an operation space and a structure space. The operation space defines the set of operations available to the architecture, and the structure space defines how the connections between operations are formed. The operation space can include different types of operations such as convolutions (2D, separable, depth-wise) and pooling (max, average); with various kernel dimensions (3x3, 5x5,7x7), strides (1,2,4), and dilation rates (1,2,3,4) [135]. The structure space could be sequential [136], [137], hierarchical [138], cell-based [135], [139], or memory-access-like topologies [140]. An example is shown in **Fig.2.3.1b *Search Space***. In the example, the operation space includes “1x1 convolution (conv)”, “3x3 conv”, “5x5 conv”, and “pool”; and the structure space is “serial” and “parallel”. The search space design often involves certain amounts of human expertise and experience.

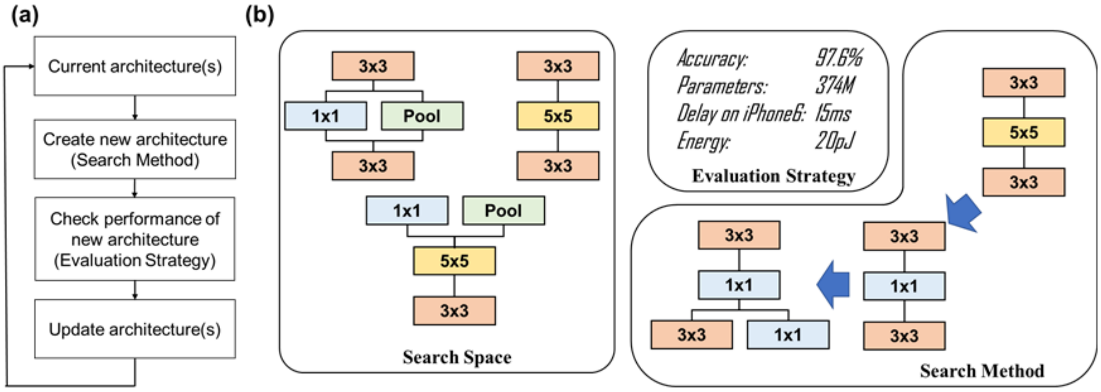


Fig.2.3.1 Illustration of a Neural Architecture Search (NAS) algorithm. (a) Flow of a NAS algorithm. The algorithm iteratively creates and evaluates new architectures to find an optimized architecture. (b) components of a NAS algorithm. The search space defines the space of all architectures. For example, the architecture could be a serial connection of a 3x3 convolution (conv), a 5x5 convolution, and another 3x3 convolution (top right in Search Space). It could branch from a 3x3conv into a 1x1 conv and a pooling in parallel, then merge back before the final 3x3 conv (top left in search space); or it could start from a parallel architecture before merging into a serial one (bottom in search space). The search method defines how the network architecture changes between each search iteration. For example, starting from the 3x3conv-5x5conv-3x3conv in the first iteration, the second layer is changed to a 1x1conv in the second iteration, and the last layer branches out another 1x1 conv in the third iteration. The evaluation strategy provides a measure of the performance of each architecture. This includes metrics such as the accuracy of the network, the memory it consumes, as well as the time and energy it takes to run.

The search algorithm defines how the network architecture changes between each search iteration. This could be re-selecting a network architecture from the search space at random or a grid-search through its parameters [141]. It could involve modification upon a previous top-performing network [138], [139]; or be determined by another neural network through a reinforcement-learning setting [136], [137]. If the architecture is differentiable, the architecture could also be trained [142]. An example of a search method is shown in **Fig.2.3.1** ***Search Method***. In the example, the search method replaces the second layer in the first iteration, and branches the third layer in the second iteration.

The evaluation strategy provides a measure of the model’s performance. This determines the NAS optimization target and is usually a combination of various performance metrics, such as accuracy, the number of flops, required memory, operation energy, latency, etc. Different designs of the evaluation strategy allow the NAS algorithm to optimize the architecture for a single metric or a trade-off between metrics under a given set of constraints. An example of an evaluation strategy is shown in **Fig.2.3.1** ***Evaluation Strategy***. In the example, we see that the accuracy, the network size (number of parameters), the delay on an iphone, and the energy are all components of the evaluation strategy. In most cases, the most important metric is an accuracy α. The traditional way to obtain α is to train and evaluate a network [143]. However, a complete train-and-evaluation procedure is costly. As a result, accuracy extraction has been accelerated through shortened training schedules on reduced-size networks [136], [139], adopting pre-trained or shared weights [144], [145], or using proxy tasks with lower complexity [135]. Accuracy could also be estimated via another neural network [140], [146].

In the remainder of this section, we present the three bio-inspired NAS algorithms used in this work. We describe the details of the search space, search algorithm, and the evaluation strategy of each algorithm. The mutation NAS algorithm is inspired by genetic mutation. During genetic mutation, the number of genes stays the same, but the genetic expressions change. Based on this observation, our mutation NAS maintains the same number of neurons, connections, and operations throughout the search process; while the architecture and the operations change. Our growth NAS is inspired by the growth of neural pathways during an organism's learning process in which new neurons and connections are formed. Taking inspiration from this, the growth NAS continuously adds neurons to the network by splitting and branching existing connections. The pruning NAS is inspired by neural pruning. During neural pruning, less important neurons are gradually removed from the network, making it more compact, efficient, and representative. Our pruning NAS follows this principle: neurons deemed less important to the task (as evaluated by an *importance* measure φ) are progressively removed during the search process. These three algorithms allow us study how the network architecture as its size (e.g. amount of neurons) is kept constant, increases, or reduces.

2.3a Mutation NAS Algorithm

Our mutation NAS is based on the search space and search method of the evolutionary algorithm described in [139] combined with the differentiable search described in [142]. The evolutionary component primarily optimizes the network structure, while the differentiable conmponent finds the optimal neurons of the network. We detail this below.

Mutation NAS Search Space

At the top level, the network of our mutation NAS has a block-based architecture (**Fig.2.3.2a**) composing an initial 3x3 convolution followed by three repetitions of ($search\ block \times N_B -$ reduction layer) stacks and a classification layer, where N_B is the number of $search\ blocks$ in each stack. The NAS optimizes the network through manipulating the architecture of each *search block*. Connections and operations in the *search block* are modified during the NAS search process, but the number of nodes stay identical.

Each $search\ block$ (**Fig.2.3.2b**) has an independent architecture composed of 8 nodes: *node0* and *node1* are replicas of the previous block outputs, *node2* to *node6* are pairwise combinations, and *node7* concatenates unused node outputs to create the *search block* output. A pairwise combination takes the output of two previous nodes, applies an operation to each, and sums the result. For example, in **Fig.2.3.2b**, *node4* takes the output of *node1* and *node2* as its inputs, applies a 3×3 operation on *node1* and a 5×5 operation on *node2*, and sums them. The available operations for a $search\ block$ (e.g., operation space) include 1×1, 3×3, 5×5, 7×7 operations and a $None$ operation.

The architecture of a $K \times K$ operation is shown in **Fig.2.3.2c**, composing a convolution with kernel size K followed by batch normalization and a *trainable neuron layer*. On the other hand, the $None$ operation passes the operation input to the operation output without applying any computation.

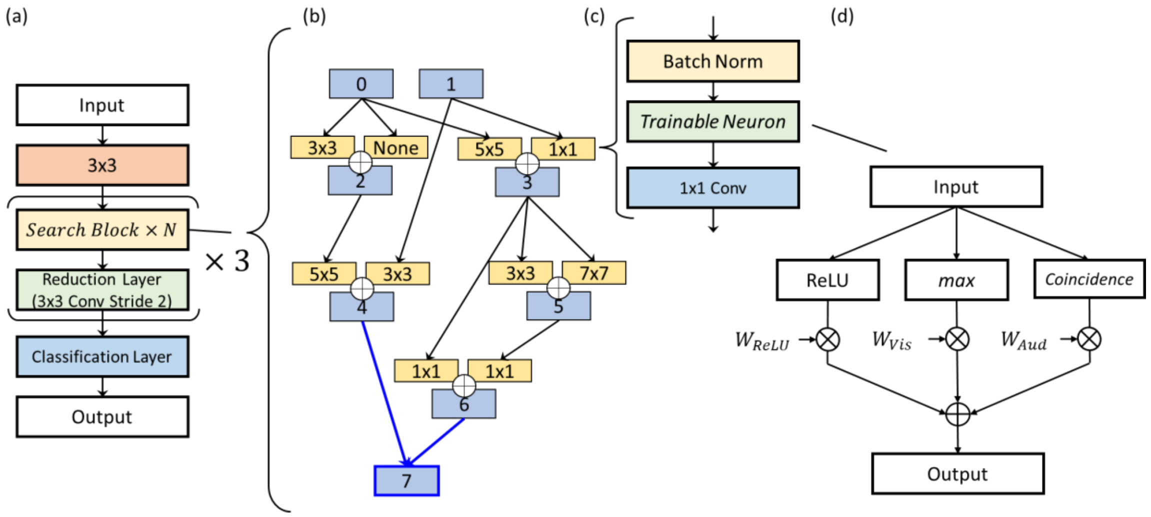


Fig.2.3.2 The network architecture of our mutation NAS. (a) The top-level schematic: a block-based architecture composing a 3x3 convolution, three ($search\ block \times N_B -$ reduction layer) stacks, and a classification layer. (b) The schematic of a *search block*: composing 8 nodes with *node0* and *node1* as prior block inputs, *node2* to *node6* being pairwise combinations, and *node7* concatenating all unused node outputs as the block output. Connections and operations in the *search block* are modified during the NAS search process, but the number of nodes stay identical. (c) The schematic of an *Operation*: a batch normalization, a *trainable neuron layer*, and a convolution with the corresponding kernel size. Note that a *1x1 Convolution* is labelled as 1x1Conv while a *1x1 Operation* is labelled as 1x1. The former refers to a single convolution with kernel size 1x1, while the latter also includes a batch normalization and a *trainable neuron layer*. (d) The *trainable neuron layer* contains three weighted paths, each corresponding to a neuron type: the conventional ReLU, the *max*, and the *coincidence* neurons. During training, the weights of the three paths are updated, and the path with the highest weight determines the neuron selected.

The *trainable neuron layer* (*TL*, **Fig.2.3.2d**) is a trainable layer that finds the optimal neuron type during training. It composes three weighted branches corresponding to the conventional ReLU neuron, the *max* neuron, and the *coincidence* neuron, respectively (e.g., the branch weights $w_{TL} = [w_{ReLU}, w_{max}, w_{co}]$ and the branch outputs $b_{TL}(x) = [ReLU(x), y_{\max\_s}(x), y_{co\_s}(x)\ ])$. A higher branch weight indicates that the corresponding neuron type has a higher probability of being the optimal neuron type. The output of the *trainable neuron layer* y_{TL} is the sum of each branch output scaled by the $softmax$ of the branch weights, e.g.,

$$y_{TL}(x) = softmax(w_{TL}) \cdot b_{TL}(x) \ldots Eq. 2.3.2$$

Where $y_{TL}(x)$ is the output of the *trainable neuron layer*. The $softmax$ of a vector is the exponential of each element in the vector divided by the sum of the exponential of all elements in the vector, e.g.,

$$softmax(z)_i = \frac{e^{z_i}}{\sum e^{z_j}} \qquad \dots Eq. 2.3.1$$

Every neuron in the *trainable neuron layer* is individually optimized. For a fully connected layer with N_F neurons, there are a total of $3N_F$ weights (e.g. 3 weights corresponding to the three neuron types for every neuron). Likewise, for a convolution layer with N_F neurons (e.g., number of output channels or feature maps), there are $3N_F$ weights.

The reduction layer reduces feature dimensions to create a more compact representation. Following the common practice [139], the reduction layer composes three components in series: (1) a 3x3 convolution with a stride (amount of shift during convolution) of 2 and a number of output channels equal to twice the amount of input channels, followed by (2) batch normalization and (3) a *trainable neuron layer*.

Mutation NAS Search Method

The search method of the mutation NAS is inspired by the genetic mutation process in an evolutionary environment. Evolution can be roughly described as follows: there exists an initial population of organisms in an environment. When some organisms (e.g., a subset of the population) meet, they compete, and the best performing organisms generate offsprings that share features of their parenst. As time passes, old and poorly performing members of the population die off. As a result, many generations later, the population will be composed of members with the best suited features for the environment. An illustration of the evolution process is shown in **Fig.2.3.3**.

Our search algorithm mimics this process. We keep an active population P composed of $n(P)$ architectures and their evaluation metrics $\epsilon_i = f(\alpha_i, \eta_i); i \in [0, n(P)]$, where α_i is the

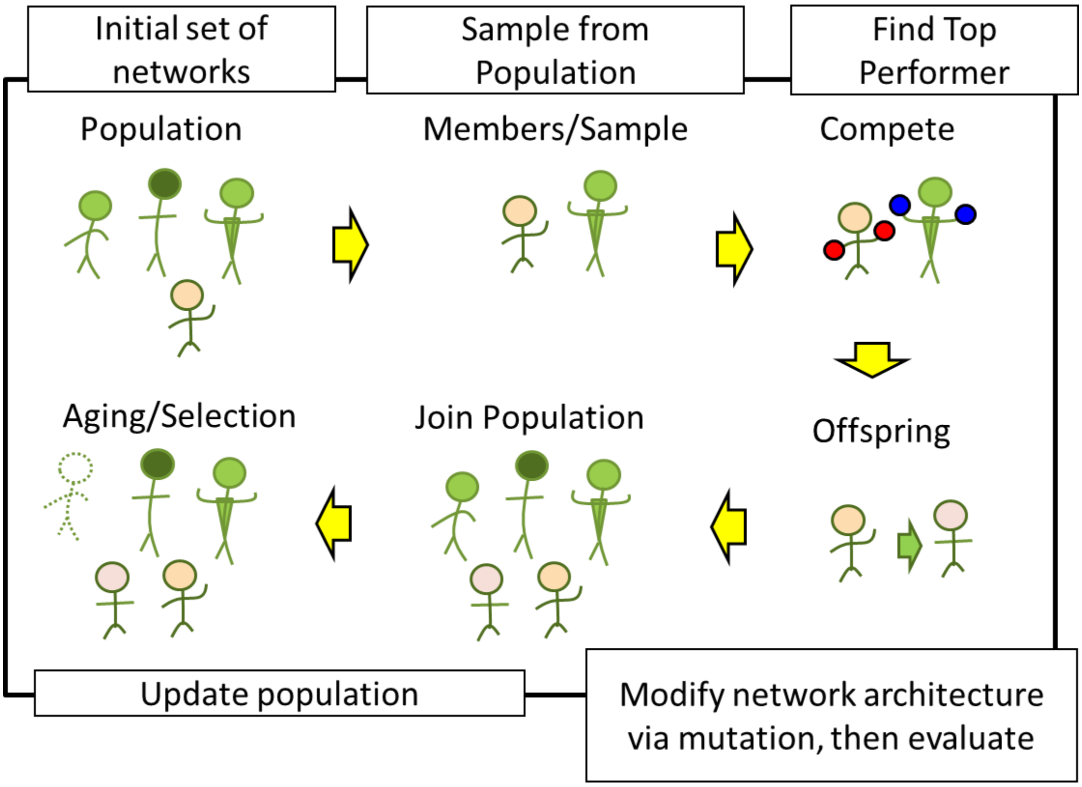
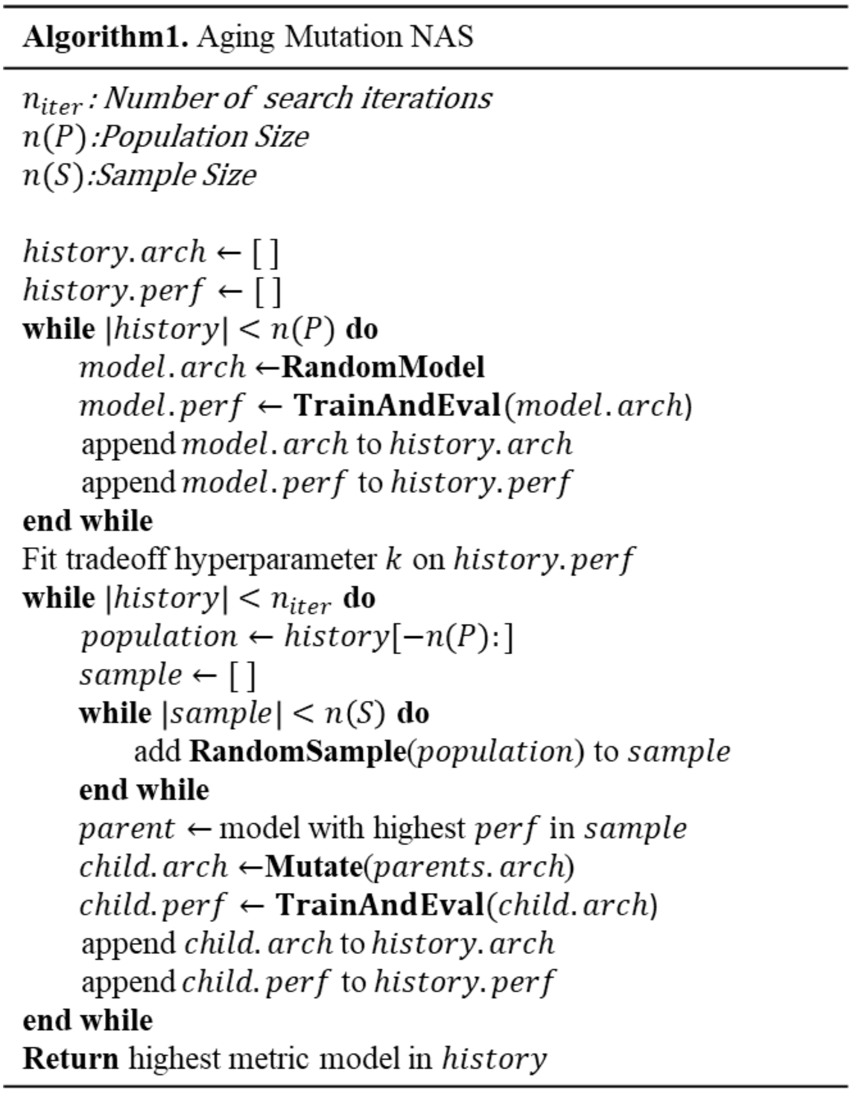


Fig.2.3.3 Illustration of the evolutionary process and the mutation NAS search method. During the search process, a population of architectures are kept. In each search iteration, architectures are sampled from the population, and the top performing architecture is selected as the parent. The parent is used to generate a child architecture by mutating its architecture. Afterwards, the child architecture is evaluated and added to the population.

accuracy of architecture i, η_i is it's number of parameters, and ϵ_i is its evaluation metric. During each search iteration, a sample S is first sampled from P. Within S, the architecture A^* with the highest evaluation metric is selected as the parent. A child A' is then generated by mutating A^*. The child is then evaluated and added to P. Finally, the oldest member (aging-evolution) or worst performing member (competition-evolution) of P is removed. The pseudo-code for the search algorithm is shown in **Fig.2.3.4**.

The mutation of an architecture is as follows: during each iteration, N_{B_S} *search blocks* are selected to mutate. For each *search block*, an *operation mutation* is applied to N_{O_S} nodes, and a *connection mutation* is applied to N_{C_S} nodes. In an *operation mutation*, one of the node's operations is randomly reselected from the operation space, as illustrated in **Fig.2.3.5a**. In this example, we see that the *None* operation in *node2* is mutated to a 7 × 7 operation. In a



```
Algorithm1. Aging Mutation NAS

$n_{iter}$: Number of search iterations
$n(P)$: Population Size
$n(S)$: Sample Size

$history.arch \leftarrow [\,]$
$history.perf \leftarrow [\,]$
while $|history| < n(P)$ do
    $model.arch \leftarrow$ RandomModel
    $model.perf \leftarrow$ TrainAndEval($model.arch$)
    append $model.arch$ to $history.arch$
    append $model.perf$ to $history.perf$
end while
Fit tradeoff hyperparameter $k$ on $history.perf$
while $|history| < n_{iter}$ do
    $population \leftarrow history[-n(P):]$
    $sample \leftarrow [\,]$
    while $|sample| < n(S)$ do
        add RandomSample($population$) to $sample$
    end while
    $parent \leftarrow$ model with highest $perf$ in $sample$
    $child.arch \leftarrow$ Mutate($parents.arch$)
    $child.perf \leftarrow$ TrainAndEval($child.arch$)
    append $child.arch$ to $history.arch$
    append $child.perf$ to $history.perf$
end while
Return highest metric model in $history$
```

Fig.2.3.4 Pseudo code of the mutation NAS algorithm. First, an initial population of architectures is generated via randomly sampling from the search space. Each architecture is evaluated, and a tradeoff parameter k is fitted on the accuracy α and parameter count η of the initial population. Afterwards, in each search iteration, we randomly sample from the population and pick the top performing architecture as the parent. The parent architecture generates a child architecture via mutation, and the child is evaluated and added to the population. The oldest member of the population is then removed. When the maximum amount of search iterations n_{iter} is reached, the algorithm returns the architecture with the highest performance metric ϵ.

connection mutation, one of the node's input connections is randomly reconnected to another node, as illustrated in **Fig.2.3.5b**. Here, we see that the left connection of *node3* is changed from *node0* to *node1*.

The parameters N_{B_S}, N_{C_S}, and N_{O_S} provide a tradeoff in exploration (e.g. trying out more diverse architectures in the search space with high N_{B_S}, N_{C_S}, and N_{O_S}) vs. exploitation

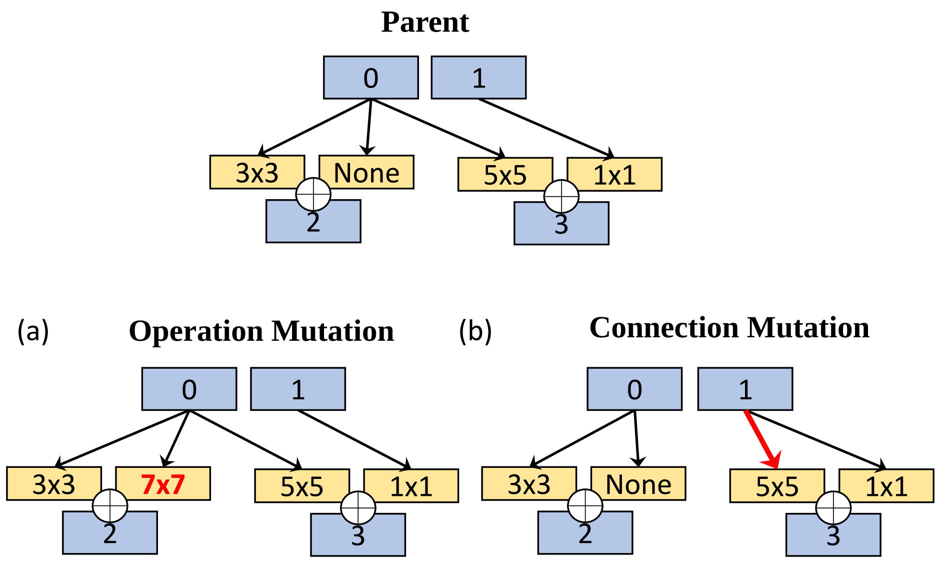


Fig.2.3.5 The mutation of an architecture. (a) The operation mutation that randomly changes one operation to another. For example, the *None operation* in *node2* is changed to a *7x7 operation.* (b) The connection mutation that randomly reconnects one input of the pairwise operation. For example, the left connection of *node3* is changed from *node0* to *node1*.

(conducting a more thorough evaluation of similar architectures in the adjacent search space with small N_{B_S}, N_{C_S}, and N_{O_S}).

Mutation NAS Evaluation Procedure

In general, the accuracy α of a neural network grows with the number of parameters η in the network. If the evaluation metric ϵ is based purely upon α, a less efficient network with higher η may outperform a more efficient network with lower η. However, under the same resource constraints, an efficient structure will have a higher α. Therefore, the network performance should be evaluated in terms of both accuracy and cost (e.g. η, flops, latency, energy, etc.). We adopt the evaluation metric $\epsilon = \bar{\alpha}/\bar{\eta}^k$, where $\bar{\alpha} = \alpha_{model}/\alpha_{init}$ is the normalized accuracy of the model, computed as the model accuracy α_{model} divided by the average accuracy of the initial population α_{init}; $\bar{\eta} = \eta_{model}/\eta_{init}$ is the normalized number of parameters in the model, and k is a hyperparameter for controlling the tradeoff between α and η obtained by fitting the relation $k = -\log\bar{\alpha} / \log\bar{\eta}$ on the initial population.

2.3b Growth-based NAS Algorithm

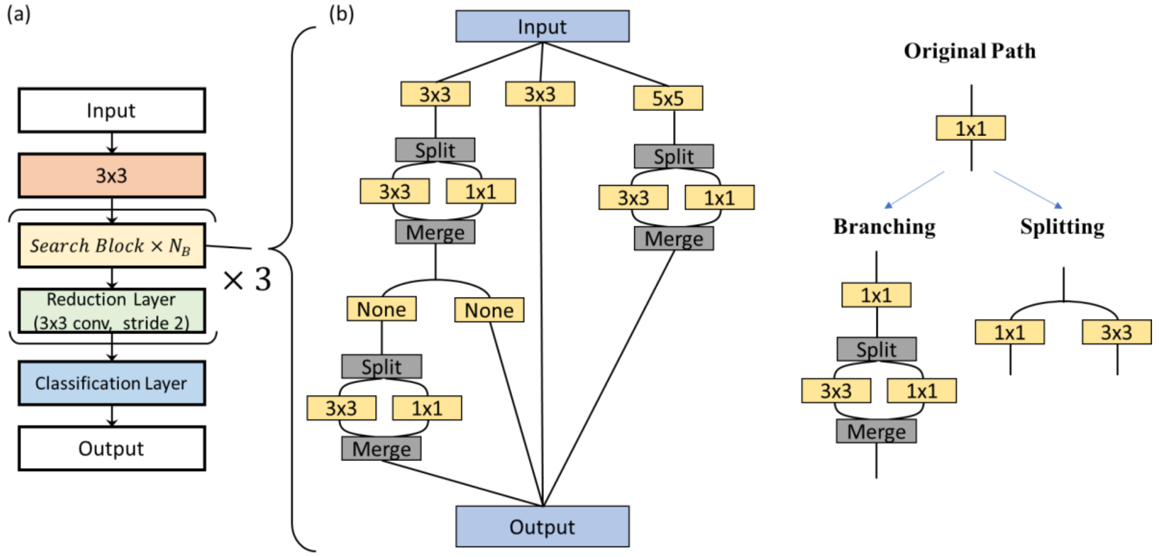
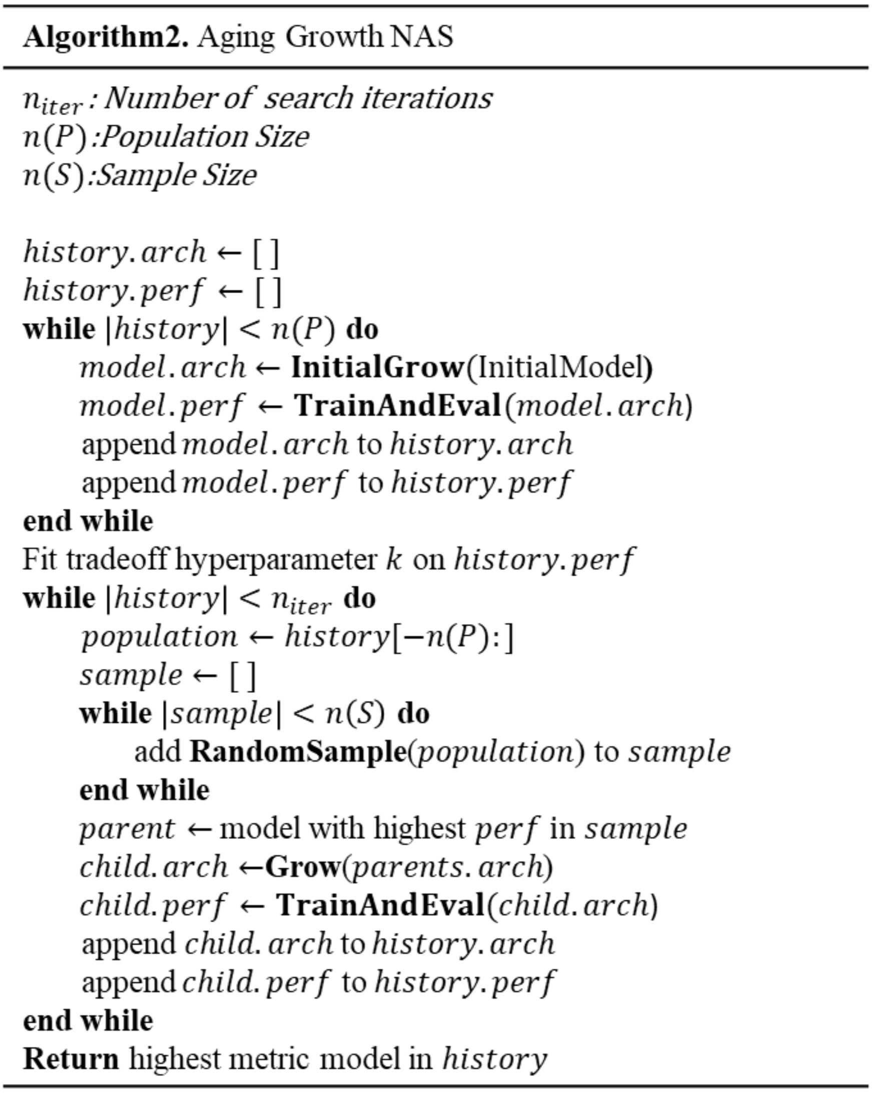


Fig.2.3.6 The network architecture of our growth-based NAS. (a) The top-level schematic: a block-based architecture composing a 3x3 convolution, three ($search\ block \times N_B -$ reduction layer) stacks, and a classification layer. (b) Schematic of a *search block*: paths are grown via *branching* and *splitting* procedures. For example, the original center path consists of only a 3×3 *operation*. The right path is created by *splitting* the center path with a 5×5 *operation*, then *branching* it with a 3×3 and a a 1×1 operation. (c) The *growth* procedure can be either a *branch* procedure or a *split* procedure. The *branch* procedure divides a path into two, applies an operation to each, and merges them back. The *split* procedure creates a second path with a different operation.

The growth NAS has a multi-branch search space [147], which is a generalized representation of modern feedforward CNN architectures [148] [149] [150]. The search method is similar to our NAS algorithm in **Sec.2.3a.** However, the procedure to generate a child architecture from a parent architecture from *mutation* process to a *growth* process.

Growth NAS Search Space

Similar to the mutation NAS, the growth NAS also has a block-based structure (**Fig.2.3.6a**) composing an initial 3x3 convolution, followed by three repetitions of ($search\ block \times N_B -$ reduction layer) stacks and a classification layer. Here, each *search block* (**Fig.2.3.6b**) is a multi-branch tree structure: neurons and paths in the block are formed via *branching* and *splitting* procedures. The *branch* procedure divides a path into two, applies an operation to each, and merges them back. The *split* procedure creates a duplicate path with a different operation. An example of a *search block* and an illustration of a *branch* and *split* procedures is shown in **Fig.2.3.6b**. In the example, the block begins from an initial 3×3 operation between the input



Algorithm2. Aging Growth NAS

```
$n_{iter}$: Number of search iterations
$n(P)$: Population Size
$n(S)$: Sample Size

$history.arch \leftarrow [\,]$
$history.perf \leftarrow [\,]$
while $|history| < n(P)$ do
    $model.arch \leftarrow$ InitialGrow(InitialModel)
    $model.perf \leftarrow$ TrainAndEval($model.arch$)
    append $model.arch$ to $history.arch$
    append $model.perf$ to $history.perf$
end while
Fit tradeoff hyperparameter $k$ on $history.perf$
while $|history| < n_{iter}$ do
    $population \leftarrow history[-n(P):]$
    $sample \leftarrow [\,]$
    while $|sample| < n(S)$ do
        add RandomSample($population$) to $sample$
    end while
    $parent \leftarrow$ model with highest $perf$ in $sample$
    $child.arch \leftarrow$ Grow($parents.arch$)
    $child.perf \leftarrow$ TrainAndEval($child.arch$)
    append $child.arch$ to $history.arch$
    append $child.perf$ to $history.perf$
end while
Return highest metric model in $history$
```

Fig.2.3.7 Pseudocode of the growth NAS algorithm. We adopt the same evolutionary backbone as the mutation NAS, e.g., keeping a population of architectures, sampling from the population, finding the top performing parent architecture, modifying the parent architecture to create a child architecture, and updating population. The main difference of the two algorithms lies in the method to generate the child architecture. Here, the child architecture is obtained by growing the parent architecture rather than mutating it. The initial population is also generated accordingly, e.g., by randomly growing an initial network.

and output (center path). The right path is then created by (1) *spliting* the center path to create the right path with a 5 × 5 operation, then (2) *branching* the right path into a 3 × 3 operation and a 1 × 1 operation.

Each operation has the same architecture as the mutation NAS in **Fig.2.3.2c** and **Fig.2.3.2d**, e.g., a convolution followed by batch normalization and a *trainable neuron layer*.

Growth NAS Search Method

The growth NAS is inspired by the growth of neural pathways: in the human brain, learning results in the creation and rearrangement of new neurons and neural connections. Upon the evolutionary backbone , we change the process to generate a child architecture from *mutation* to *growth*. From the parent architecture, N_{B_S} *search blocks* are randomly selected, and N_{G_S} random *growth* procedures, (e.g., random *branch* or *split* procedures) are applied to each block.

For the initial population, we begin with a single 3×3 operation between the input and output of each *search block*. Each block then randomly *grows* (*branches* or *splits*) up to N_{G_Init} times. The pseudocode of the growth NAS search algorithm is shown in **Fig.2.3.7**.

Growth NAS Evaluation Procedure

The growth NAS optimizes for the same target as the mutation NAS, e.g., a tradeoff between the accuracy and the cost via the evaluation metric $\epsilon = \bar{\alpha}/\bar{\eta}^k$,

2.3c Pruning NAS Algorithm

The pruning NAS begins with arguably the most widely benchmarked hand-crafted architecture: the ResNet [149]. ResNet introduced skip connections, enabling significantly deeper networks and marking a drastic improvement in deep learning performance across a large variety of tasks. We start with a ResNet that contains an equal amount of each neuron type, then consecutively remove low-importance neurons until we arrive at a high-accuracy, efficient network.

Pruning NAS Search Space

The top level architecture of a ResNet is shown in **Fig.2.3.8a**, composing a 3x3 convolution, three ($residual\ block \times N_B -$ reduction layer) stacks, a global average pooling layer, and a classification layer. The architecture of a *residual block* is illustrated in **Fig.2.3.8b**, composing two consecutive 3×3 operations and a skip connection from the block input to the block output. The skip connection provides a shortcut for signals and gradients between the block input and output.

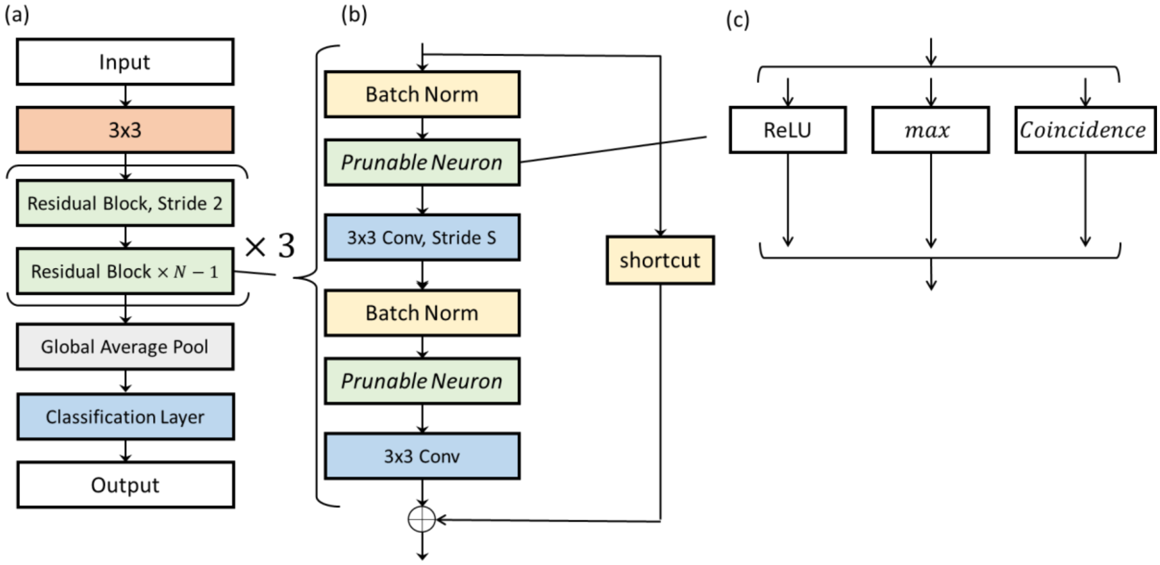
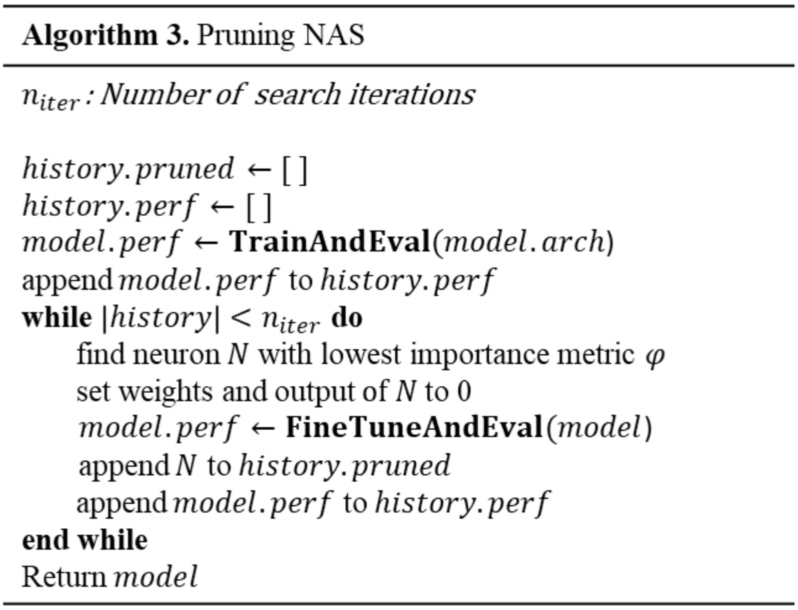


Fig.2.3.8 The network architecture of our pruning NAS. (a) The top-level schematic of a ResNet, composing an initial 3x3 layer, three ($residual\ block \times N_B$ − reduction layer) stacks, a global pooling layer, and a classification layer. (b) The schematic of a *residual block*: two consecutive 3 × 3 operations and a skip connection (shortcut) from the block's input to its output. The 3 × 3 operation follow a similar implementation as the mutation and growth NAS, but with a *prunable neuron layer* instead of a *trainable neuron layer*. e.g., batch normalization→*prunable neuron layer* → 3x3 convolution. (c) A *prunable neuron layer* composes an identical amount of each neuron type (e.g., 1/3 being the ReLU, *max*, and *coincidence* neurons, respectively).

Instead of a *trainable neuron layer*, each operation is built upon a *prunable neuron layer*. A *prunable neuron layer* allocates 1/3 of it's neurons to each neuron type. During pruning, the neurons in the *prunable neuron layer* will be gradually removed based on their importance.

Pruning NAS Search Method

The pruning NAS is clearly inspired by neural pruning: as the human brain is becoming more skilled at a task, neurons and connections are gradually removed to create a more mature and efficient network. Our pruning NAS (**Fig.2.3.9**) follows this procedure: we begin with a fully trained network composing an equal amount of each neuron type; then, in each iteration, we remove N_{R_S} neurons with the lowest *importance metric* φ. The *importance metric* provides an estimation of how much the neuron contributes to the network's operation. When neurons with low *importance metric* are removed, we expect that it does not significantly affect the network's operation. A neuron is removed by setting its weights and outputs to 0. Afterwards,



Algorithm 3. Pruning NAS

```
$n_{iter}$ : Number of search iterations

$history.pruned \leftarrow [\,]$
$history.perf \leftarrow [\,]$
$model.perf \leftarrow$ TrainAndEval($model.arch$)
append $model.perf$ to $history.perf$
while $|history| < n_{iter}$ do
    find neuron $N$ with lowest importance metric $\varphi$
    set weights and output of $N$ to 0
    $model.perf \leftarrow$ FineTuneAndEval($model$)
    append $N$ to $history.pruned$
    append $model.perf$ to $history.perf$
end while
Return $model$
```

Fig.2.3.9 Pseudocode of the pruning NAS. We first train a network that has an identical amount of each type of neuron. In each search iteration, the neuron with the *lowest importance metric* φ is removed. After a neuron is removed, the network is fine-tuned via a shortened training process. The search process ends when the maximum amount of iterations is reached.

the network is fine-tuned via a shortened training procedure with a reduced learning rate. This process continues until the maximum number of iterations n_{iter} is reached or when the accuracy of the network drops by a predetermined amount.

Several *importance metrics* have been previously proposed, such as considering neurons with low weights as less important [151]; measuring how much the neuron contributes to the loss function [152]; or how much it contributes to the creation of features in subsequent layers [153]. Here, we adopt the squared weight *importance metric*, e.g., the *importance metric* of a neuron is the average of the neuron's squared weights ($\varphi = \overline{w^2}$).

2.4 Experiment Setup

In this section, we describe the setup for the NAS experiments. We will first describe the datasets used along their data characteristics (i.e., dimensions, number of samples, classes, etc.) and how the data is prepared for training (i.e., normalizing, cropping, etc.). We will then describe the settings/parameters of the NAS algorithm (i.e., N_{B_S}, N_{O_S}, N_{G_S}, $n(P)$, $n(S)$, etc.)

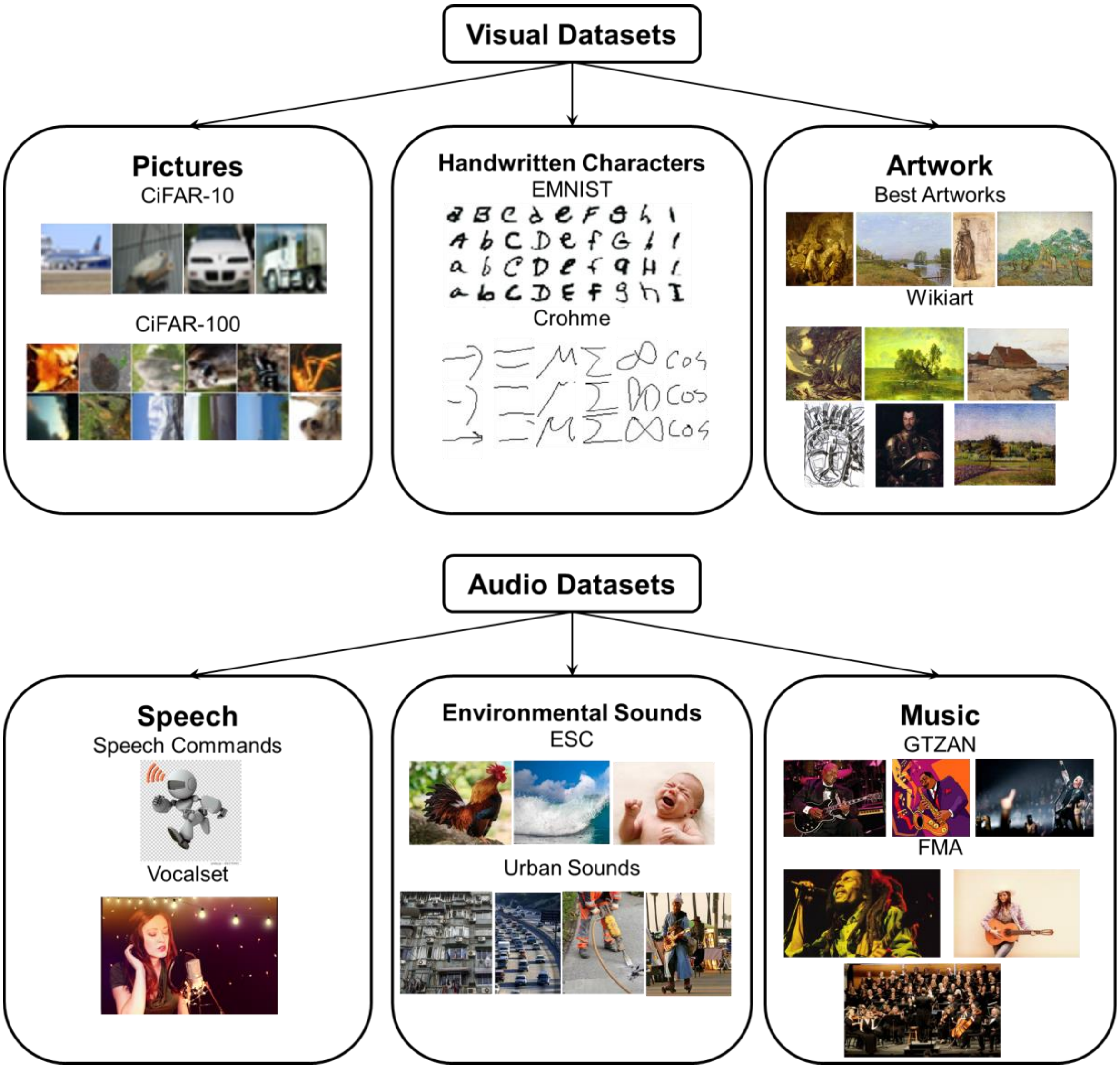


Fig.2.4.1 List of datasets used in this research. In the visual recognition category, the subcategories include pictures, handwritten characters, and artwork. In the audio recognition category, the subcategories include speech, environmental sounds, and music. For every subcategory, we select two datasets to experiment on.

as well as the training and evaluation of the networks (i.e. learning rate schedule, number of epochs, etc.).

2.4a Data Preparation

The hierarchy of datasets used in this work is shown in **Fig.2.4.1.** There are two categories of tasks, each with three subcategories, and each with two datasets. For visual recognition, the subcategories are pictures (CiFAR-10, CiFAR-100 [154]), handwritten characters (EMNIST [155], CROHME [156]), and artwork (Best Artworks [157], Wikiart [158]). For audio recognition, the subcategories are speech (Speech Commands [159], Vocalset [160]), environmental sounds (ESC [161], Urbansounds8k [162]), and music (GTZAN [163], FMA [164]). In the remainder of this section, we describe in detail each dataset in terms of their content, data characteristics, and preprocessing steps for training.

2.4a.1 Visual Recognition Datasets

Pictures: CiFAR-10 and CiFAR-100

The CiFAR datasets [154] named after the Canadian Institute For Advanced Research contain 60,000 colored, 32x32 images split into a training subset of 50,000 and an evaluation subset of 10,000. The CiFAR-10 dataset has 10 classes containing objects and animals like airplanes, trucks, cars, and dogs. On the other hand, CiFAR-100 has 100 classes that contain a larger variety of classes such as humans, plants, and natural/man-made objects. Some examples include cloud, boy, woman, roses, sunflowers, bottles, etc.

For the CiFAR datasets, we follow the common practice of normalizing the image by the mean and average of each channel (e.g., red, green, and blue). We do not apply any other form of data augmentation (manipulating the images to improve generalization by flipping, cropping, rotation, brightness, and other means).

Handwritten Characters: EMNIST

The Extended MNIST (EMNIST) dataset [155] is an extension of the MNIST dataset derived from the NIST Handprinted Forms and Characters Database. There are ~700k training and ~100k testing images in the greyscale, 28x28 pixel format. The dataset contains a total of 47 classes, including digits (0~9) and letters (A~Z).

For the EMNIST dataset, we follow the standard greyscale-image preprocessing practice of standardizing the image to the [0,1] range. Again, we do not use any additional form of data augmentation.

Handwritten Characters: CROHME

The Competition on Recognition of Online Handwritten Mathematical Expressions (CRHOME) dataset [156] contains ~300k training and 75k test images of hand-written mathematic symbols. The images are presented in a 45x45 pixel greyscale format. There are 81 classes, which include numbers, Greek letters (ex. α, β, γ), variables (ex. x, y, w, z), parenthesis (ex. (),[],{}), mathematical operators (ex. $+, -, \times, \div$), and other expressions.

Similar to the EMNIST dataset, we standardize the greyscale images to the [0,1] range. We do not use any additional form of data augmentation.

Artwork: Best Artworks

The Best Artworks of All Time dataset [157] is a challenge organized on Kaggle (an online machine learning competition platform) whose goal is to recognize paintings from influential artists. There is a total of ~9,000 color images with dimensions ranging from a few hundred to two thousand pixels. The 50 classes correspond to artists, including Vincent Van Goh, Pablo Picasso, Leonardo Da Vinci, etc.

As there is no predefined training and evaluation subset, we divide the 9,000 images into training and evaluation subsets via an 80-20 split. In other words, 80% of the images are taken at random to be the training set and the remaining 20% of the images as the evaluation set. Also, since the images vary in resolution, we take 5 random 64x64 crops for every image and treat each as an individual example, resulting in a training set of 35,000 examples and 8,500 evaluation examples. Otherwise, no additional form of augmentation is applied.

Artwork: Wikiart

The Wikiart dataset [158] contains artwork from a variety of genres, artists, and styles. There is a total of ~80,000 color images, each of which has dimensions between a few hundred to four thousand pixels. There are 27 classes that correspond to specific art styles such as Impressionism, Realism, Symbolism, Early Renaissance, etc.

Similar to the Best Artworks dataset, we split the images into training and evaluation subsets via an 80-20 split. 5 random 64x64 crops are taken from each image and treated as individual examples, resulting in a training set of 325k examples and 80k evaluation examples.

2.4a.2 Speech Recognition Datasets

Speech: Speech Commands

The speech commands dataset [159] contains spoken keywords of robotic commands. There is a total of ~60k recordings split into a training set of ~50k and an evaluation set of ~10k. Each recording has a duration of ~1 second. There are 30 classes that include the commands Yes/No, up/down/left/right, on/off, numbers, and other phoneme-rich words.

To process each recording, we first clip or zero-pad each recording to a 1-second duration. We then compute its Mel-frequency spectrogram, which is a short-term power spectrum computed on a nonlinear frequency scale that approximates the human auditory system's response. The spectrogram is computed with 32 channels using a frame size of 512 points and a hop length (number of points between successive frames) of 256 points. A small value of 0.00001 is then added to the spectrogram before taking the log of the spectrogram to avoid log(0) conditions. Finally, the log-spectrogram is linearly interpolated to a dimension of 32x32 and normalized.

All other audio datasets follow this log-Mel-spectrogram approach, with the number of channels and the final spectrogram dimensions scaled proportionally to the length of the recording.

Speech: Vocalset

The Vocalset [160] is a dataset composed of singing techniques performed by professional singers. There is a total of 3,500 recordings in the dataset, each up to 10 seconds long. The 17 classes correspond to singing techniques like belting, vibrato, vocal fry, inhaled, etc.

We preprocess the Vocalset similar to that of Speech Commands: cropping or zero-padding each recording to 5 seconds length, computing a 128-channel log-Mel-spectrogram, then interpolating the spectrogram to 128x128 and normalizing it.

Environmental Sounds: ESC

The Environmental Sound Classification (ESC) dataset [161] is a collection of environmental sounds extracted from the Freesounds project. There are 2000 recordings, each 5 seconds long. The dataset is prearranged into 5 folds. This means that every 20% of the data is organized into a group called a “fold”. This arrangement allows different users to cross-validate their results using the same examples while allowing flexibility in the amount of examples in the test and evaluate split. Each recording is classified into one of 50 classes belonging to 5 major categories: Animals (dogs, cats, cows, etc.), Natural soundscapes (rain, sea waves, etc.), Human sounds (crying, coughing, etc.), Interior/domestic sounds (door creak, can opening, etc.), and Exterior/Urban sounds (helicopter, train, etc.).

We use the first to fourth folds are for training and the fifth for evaluation. As previously mentioned, the recordings are processed by padding/clipping to 5s, computing the 128-channel log-Mel-spectrogram, interpolating to 128x128, and normalizing it.

Environmental Sounds: UrbanSounds8k

The Urbandsounds-8k dataset [162] contains 8732 excerpts of urban-environment sounds extracted from the Freesounds project. Each clip is up to 4 seconds long and labeled as one of 10 classes that include car horns, dog barking, sirens, street music, etc. The dataset is prearranged into 10 folds.

We use the first to eighth folds for training and the ninth to tenth folds for evaluation. The recordings are processed by padding/clipping to 3 seconds length, computing the 64-channel log-Mel-spectrogram, interpolating to 64x64, and normalizing it.

Music: GTZAN

The GTZAN dataset [163] named after its organizer George Tzanetakis is a music genre dataset with a total of 1,000 recordings. Each recording is 30 seconds long, and there are 10 classes corresponding to music genres including blues, classical, country, disco, rock, etc.

We again divide the dataset into training and evaluation subsets via an 80-20 split. Afterward, each 30-second recording is divided into six 5-second clips, and each clip is processed by computing the 128-channel log-Mel-spectrogram, interpolating to 128x128, and normalizing it. This results in a training set of 4,800 and an evaluation set of 1,200.

Music: FMA Medium

The Free Music Archive (FMA) dataset [164] is a large-scale music dataset that contains tracks from creative-commons-licensed audio. Data is available as pre-computed features, 30-second recordings, as well as full-length audio. For each recording, the title, artist, album, and genre are provided. The FMA medium subset has 25,000 30-second recordings; and is labeled as one of 16 music genres that include blues, jazz, pop, etc.

Similar to the GTZAN dataset, we use an 80-20 train-eval split and divide each 30-second recording into six 5-second clips. Each clip is processed by computing the 128-channel log-Mel-spectrogram, interpolating to 128x128, and normalizing it. This results in a training set of 120,000 and an evaluation set of 30,000.

2.4b NAS and Training Setup

During NAS, it is important to adopt a suitable model size (i.e., layers, number of neurons per layer, etc.) and training setting to avoid overfitting and underfitting. When networks are overfitted, their performance becomes capped independent of the network architecture with random fluctuation between different iterations. When networks are underfitted significantly, the non-converged, large variation in accuracy also it difficult to compare architectures. Both overfitting and underfitting would misguide the search process.

In this section, we describe in detail the NAS search algorithm settings (i.e., search iterations, N_{G_S}, N_{B_S}, N_{O_S}, $n(P)$, etc.), network settings (number of filters $n_F$and blocks N_B), and the training/ evaluation settings (learning rate schedule, loss functions, number of epochs, etc.) during the search process.

Mutation-based NAS

Finding optimal NAS settings for every dataset would be extremely time-consuming. Thankfully, we can modify and build upon the setting of similar prior works as NAS on the CiFAR-10 dataset is well benchmarked. For this, we utilize theoretical and experimental studies on the scaling of network size and training settings with respect to the dataset size[165].

The mutation-based NAS search algorithm requires determining the population size $n(P)$, sample size $n(S)$, number of neurons/filters n_f and blocks N_B in the network, the search iterations n_{iter}, and the number of mutation blocks N_{B_S}, connections N_{C_S}, and operations N_{O_S} in each search iteration. In a prior mutation-based NAS [139], the authors experimented with a variety of population $n(P)$ and sample sizes $n(S)$ and achieved best results via a population size $n(P) = 100$ and sample size $n(S) = 25$; using a network size of $n_f = 24$. The NAS improvement was observed to plateau at $20k$ iterations, e.g., $n_{iter} = 2 \times 10^4$. We keep the same $n(P)$ and $n(S)$; however, since our operation space is limited to convolutions of different sizes (while the reference includes 3 different types of convolutions), n_{iter} is reduced to 10^3 and n_f is reduced to 16 to further avoid overfitting. Finally, as our goal is to study characteristics of high-performing networks containing specialized neurons rather than obtain a global optimized network, we focus on exploitation over exploration: N_{B_S}, N_{C_S}, and N_{O_S} are all set to 1. The NAS search algorithm settings are summarized in **Table2.4.1**.

The training settings for each search iteration include: the number of training epochs n_e (or training steps n_s), the gradient descent method, the learning rate schedule, the batch size n_{batch},

Table 2.4.1 Training settings for each dataset.

Dataset	CiFAR-10	CiFAR-100	EMNIST	Crohme	Paintings	Wikiart
Input Dims.	32x32x3	32x32x3	28x28x1	40x40x1	64x64x3	64x64x3
Classes	10	100	10	81	50	27
Examples	50,000	50,000	700,000	300,000	7,000	64,000
Depth	9	9	9	9	9	9
Width	16	16	20	14	6	18
Epochs	10	10	3	4	26	8

Dataset	Speech	Vocalset	ESC	Urban	GTZAN	FMA
Input Dims.	32x32x1	128x128x1	128x128x1	64x64x1	128x128x1	128x128x1
Classes	30	17	50	10	10	16
Examples	85,000	4,000	2,000	8,000	1,000	25,000
Depth	9	9	9	9	9	9
Width	20	4	4	6	4	12
Epochs	8	38	50	25	70	14

and the weight decay. Following the previous work [139], we use stochastic gradient descent with a momentum of 0.9 as the gradient descent method. The initial learning rate is set to 0.01 and decreased by 80% for every 30% of the total number of training steps (e.g., reduced to 2×10^{-3} at the 30th percentile step, 4×10^{-4} at the 60th percentile step, and 8×10^{-5} at the 90th percentile step). The batch size is 128 and weight decay is 5×10^{-4}. However, we reduce the number of training epochs from 25 to 10 as our network is smaller. To accelerate the neuron-type search, we also increase the *trainable neuron layer* weights w_{TL} by $10\,\times$.

Using CiFAR-10 as a baseline, we scale the training settings for each dataset based on the results reported in [165]. Empirically and theoretically, the authors suggested that network size and training steps should be proportional to the square root of the number of training samples in the dataset. In addition, we also consider the increase in input complexity that comes from the number of input channels:

$$n_{f,DS} = n_{f,Cifar}\sqrt{(N_{DS}/N_{Cifar})\frac{C_{DS}}{C_{Cifar}}} \quad \dots Eq. 2.4.1$$

Where for a given dataset (DS), N_{DS} is the number of training samples in the dataset, C_{DS} is the number of channels of each sample, and $n_{f,DS}$ is the number of neurons/filters of the network used in NAS. The reference dataset, CiFAR-10, is denoted with subscript $Cifar,$ e.g., N_{Cifar}, C_{Cifar}, and $n_{f,Cifar}$. Likewise,

$$n_{s,DS} = n_{s,Cifar}\sqrt{(N_{DS}/N_{Cifar})} \quad \dots Eq. 2.4.2$$

Where $n_{s,DS}$ denotes the number of training steps for a dataset and $n_{s,Cifar}$ is the number of training steps for the reference CiFAR-10 dataset. Note that n_s is related to n_e via the relation $n_s = n_e \times N_{DS}/n_{batch}$. The training settings for each dataset are summarized in **Table2.4.1**.

Growth-based NAS Algorithm

Due to sharing the same evolutionary backbone, the growth-based NAS shares many of the search algorithm parameters as the mutation-based NAS. Specifically, $n(P) = 100$, $n(S) = 25$, $n_{iter} = 1000$ and $n_f = 16$. The initial population is generated by randomly applying up to $N_{G\_Init} = 10$ growth operations to each block. In each search iteration, the number of blocks that grow N_B and the number of growth operations per block N_{G_S} are both set to 1. In addition, to prevent networks from overgrowing, we also terminate the search algorithm when the total number of *operations* in the network exceeds 100.

Pruning-based NAS Algorithm

The search algorithm setting is as follows: the number of neurons pruned $N_{R\_S} = 1$, $n_{iter} = 1000$, and $n_f = 16 \times 3$ (e.g., $n_f = 16$ for each neuron type: conventional, *max*, and *coincidence*). In addition, as the network begins to fail when too many neurons are removed, we terminate the search algorithm when the network recognition rate drops by 2%

The initial network is trained using the same settings as the mutation and growth NAS, but with $50 \times$ the number of steps n_s. This corresponds to $n_e = 500$ epochs with stochastic gradient descent with a momentum of 0.9, an initial learning rate of 0.01 decreased by 80% for

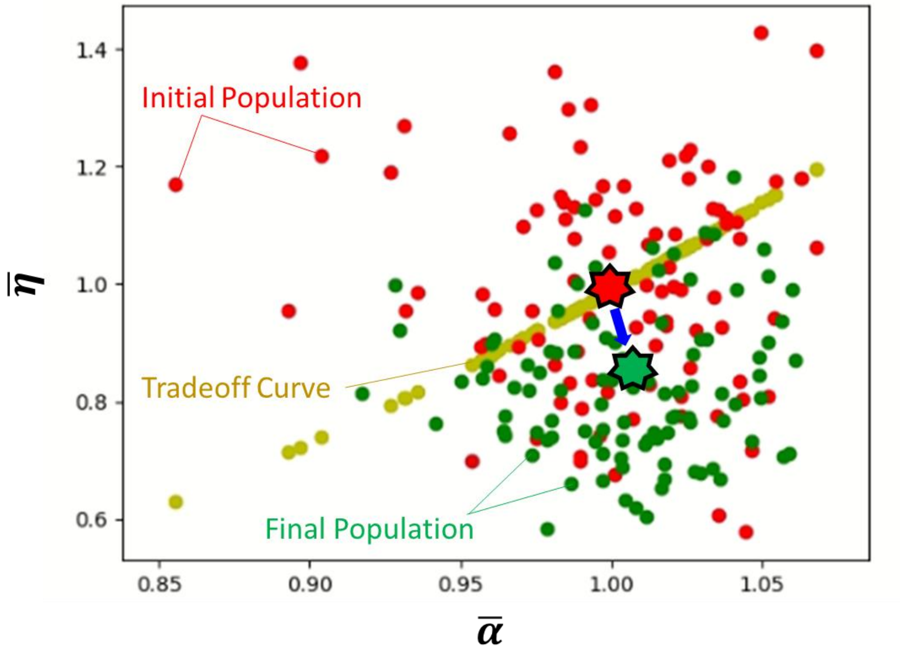


Fig.2.5.1 Scatter plot of network performance ($\bar{\eta}$, $\bar{\alpha}$) of the initial (red) and final (green) population of the mutation NAS on the CiFAR-10 dataset. The line of tradeoff $\bar{\alpha} = \bar{\eta}^k$ is displayed in yellow. The NAS successfully searched and obtained high performance networks to the right-bottom of the tradeoff line; with the final population having 1% higher $\bar{\alpha}$ and 18% lower $\bar{\eta}$.

every 30% of the total number of training steps, $n_{batch} = 128$ and weight decay is 5×10^{-4}. The fine-tuning process composes two epochs, each with the decayed learning rate (2×10^{-3}) and the double-decayed learning rate (4×10^{-4}), respectively.

2.5 Experimental Results

In this section, we analyze three characteristics of top-performing networks: (1) the *structural organization* of the network, e.g., the height (or depth) and width of the *search blocks* as the network progresses; (2) the *structural composition* of the network, e.g., the type and number of *operations* in the *search blocks* as the network progresses; and (3) the *neural composition* of networks, e.g., the amount of each type of neurons in the search blocks as the network progresses.

We study the *structural organization* and *structural composition* on networks optimized via the mutation and growth NAS, as they have diverse operations and connections during the

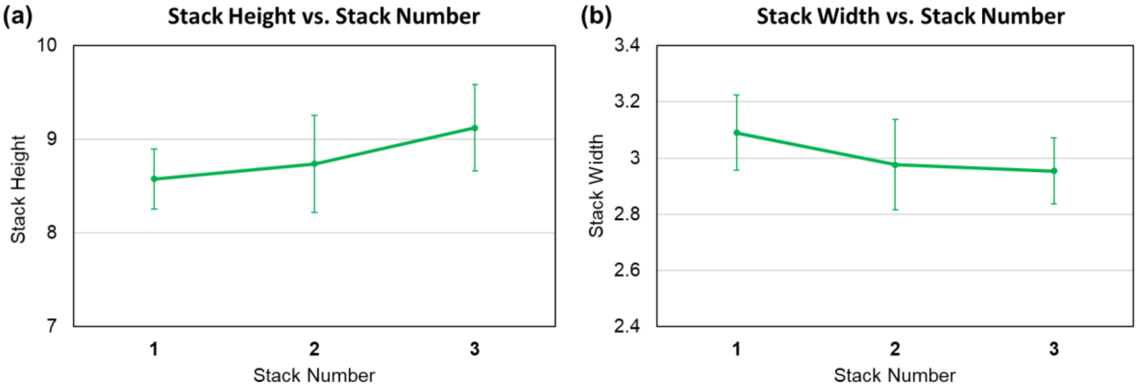


Fig.2.5.2 *Structural organization* of the top performing networks discovered by the mutation NAS. (a) stack height, and (b) stack width. Towards the end of the network (latter stacks), the height continues to increase while the width decreases. This is due to the cost tradeoff: by decreasing width, the network cost reduces. Under the constraint of fixed number of neurons, this squeezes the architecture to a greater height

search process. On the other hand, we study the *neural composition* on the pruning NAS; which removes the impact from different types of operations and connections.

Fig.2.5.1 shows a scatter plot of the networks optimized by the mutation NAS on the CiFAR-10 dataset. The x and y axes represent the normalized accuracy $\bar{\alpha}$ and the normalized number of parameters $\bar{\eta}$, respectively. The initial population is displayed in red and the final population in green. The $\bar{\alpha} - to - \bar{\eta}$ tradeoff curve ($\bar{\alpha} = \bar{\eta}^k$) is displayed in yellow. As the architecture search progresses, the NAS obtains networks at the bottom right corner below the tradeoff line, e.g., networks with higher performance but a lower number of parameters. Our NAS successfully achieves this goal with the final population (green star) displaying a 1% higher $\bar{\alpha}$ (72% →73%) and 18% lower $\bar{\eta}$ (4.8M →4M) compared to the initial population.

Fig.2.5.2 shows the *structural organization* of top-performing networks found by the mutation NAS. Here, we report statistics for each stack in **Fig.2.3.2a** (e.g., every N_B *search blocks*). The height H of an architecture is defined as the longest path from its input to output. For example, the *search block* in **Fig.2.3.2b** has a maximum height of 4, via the path through *node0*→*node3*→*node5*→*node6*→*node7*. The width W of an architecture is defined as the maximum number of *operations* of the same height (or equivalently, the maximum number of

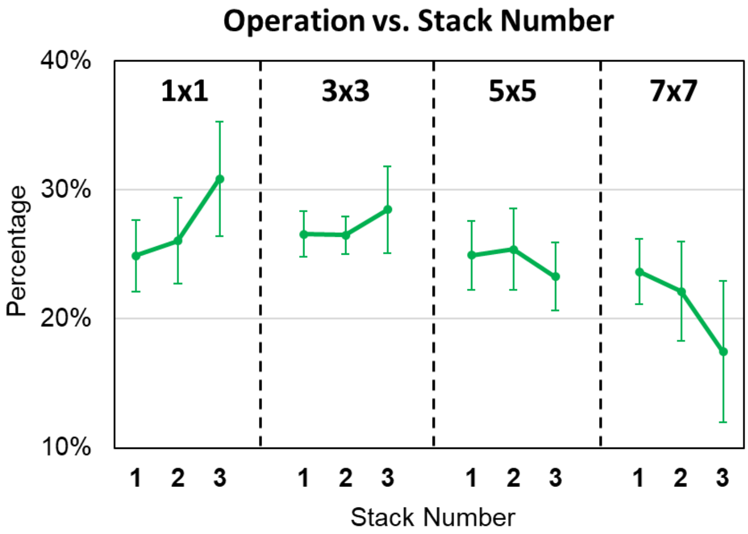


Fig.2.5.3 *Structural composition* of top performing networks found by the mutation-based NAS. The percentage of 1 × 1, 3 × 3, 5 × 5, and 7 × 7 *operations* in each stack are shown. In the first stack, the percentage of each operation is similar. Towards the end of the network, a significant decrease in the higher kernel size (K) operations is observed, as expected from the cost tradeoff.

neurons or filters). For example, the structure in **Fig.2.3.2b** has a width of 2, as *node4* and *node5* both share a height of 2.

We observe the networks display increasing height and decreasing width as the network progresses towards the latter stacks in **Fig.2.3.2a**. This is due to the cost tradeoff: the cost (e.g., η) of an *operation* is proportional $n_{in} \times n_{out} \times K^2$, where n_{in} is the number of input filters, n_{out} is the number of output filters, and K is the kernel size. As a result, the cost of a network can be reduced by either (design.1) adopting a structure with a lower width, which reduces the overall n_{in}; (design.2) avoiding operations in the latter stacks which have both larger n_{in} and n_{out}; and (design.3) adopting operations with smaller kernel size K. It is obvious that (design.1) is observed in **2.5.2(b).** As the mutation NAS keeps a constant number of operations and connections, this results in a greater height as observed in **2.5.2(a).**

Fig.2.5.3 shows the *structural composition* of the top-performing networks. We observe that, in the first stack, the 1x1, 3x3, 5x5, and 7x7 *operations* occupy a similar percentage. Towards the end of the network, the number of operations with a small kernel size (K) increases while

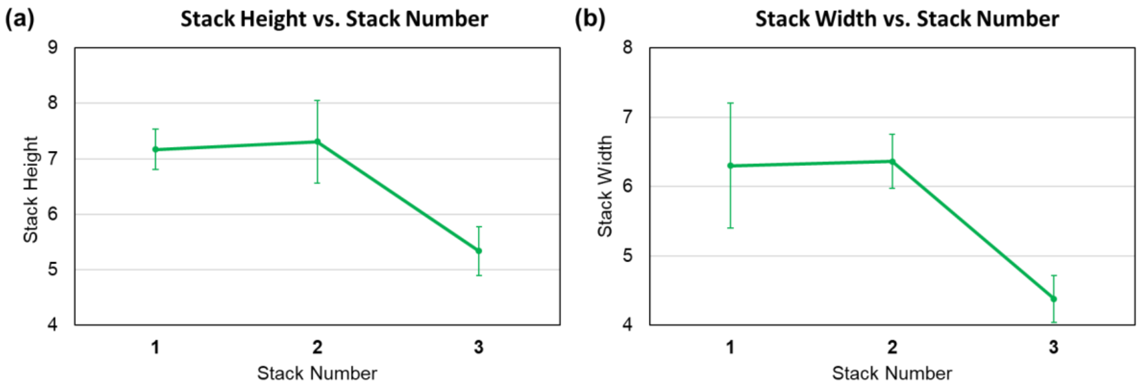


Fig.2.5.4 *Structural organization* of the top performing networks discovered by the growth NAS. (a) stack height, and (b) stack width. Both the height and width drop significantly at the 3rd stack, as a result of the cost tradeoff: a small 3rd block reduces the network cost. From the 1st to 2nd stack, on the other hand, while the average height and width both increase, we observe cases both in the increase of height/width and the decrease of height/width. This could be a compensation for the extreme reduction in 3rd block size.

the percentage of operations with large K reduces. This is a characteristic of (design.2). In earlier blocks, having a more diverse K provides an image-pyramid-like representation that can view the input image at different scales. However, operations with large K become too expensive at the latter blocks, and thus their presence reduces.

2.5b Growth-based NAS

While the mutation NAS has a fixed number of operations in each block, the growth NAS can grow indefinitely until the tradeoff between α and η is unfavorable. **Fig.2.5.5** shows the *structural organization* of the top-performing networks found by the growth NAS. We observe that the stack height and width both drop drastically in the third block, characteristic of (design.1). The average width and height increase slightly from the 1st to 2nd block; however, cases of width/height decrease are also observed. This increase could be to compensate for the significantly smaller 3rd block, e.g., operations originally in the 3rd block can be appended to the 2nd instead, which, for the same operation, reduces n_{in} and n_{out}.

Fig.2.5.6 shows the *structural composition* of the top-performing networks found by the growth-based NAS. We observe that the 3×3 *operation* stands out with the highest

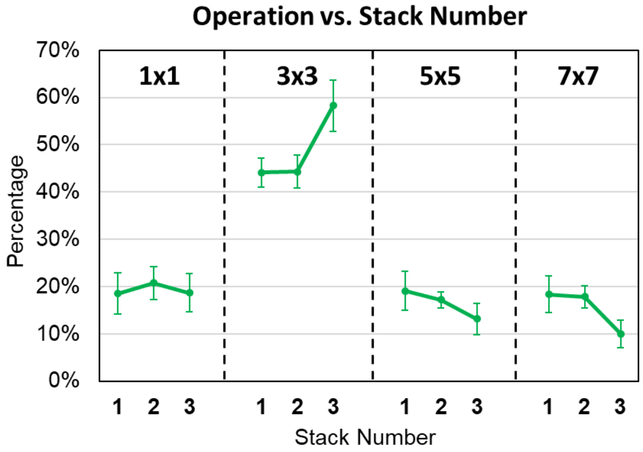


Fig.2.5.7 *Structural composition* of top performing networks found by the growth-based NAS. The percentage of 1 × 1, 3 × 3, 5 × 5, and 7 × 7 *operations* in each stack are shown. We observe that the percentage of 3 × 3 stands out with having a good cost-to-performance tradeoff. Otherwise, a similar trend to the mutation NAS is observed, e.g., a a significant decrease in the percentage of higher kernel size (*K*) operations towards the end of the network.

percentage. This indicates that the 3 × 3 *operation* provides a very good cost-to-performance tradeoff, which agrees with prior hand-designed networks [38]. We also observe the same trend as the mutation NAS, where an image-pyramid representation was observed in the 1st stack and the percentage of high-*K* operations decreasing towards the end of the network.

2.5c Pruning-based NAS

The pruning-based NAS does not change the operations and the connections throughout its search process. While this impedes its ability to study *network structure* and *network composition*, it provides an advantage in studying the *neural composition* in terms of the importance, location dependency, and quantity as it takes out structural contributions.

Fig.2.5.8 shows the importance of neurons in the network as pruning progresses. Network layers are represented in strips, from the earlier layers on the top to the latter layers on the bottom. In each strip, the conventional (ReLU), *max*, and *coincidence* neurons are represented by red, green, and blue, respectively, while neurons pruned are displayed in black. Neurons are ranked by their importance metric φ from the left (high φ) of each strip to the right (low φ).

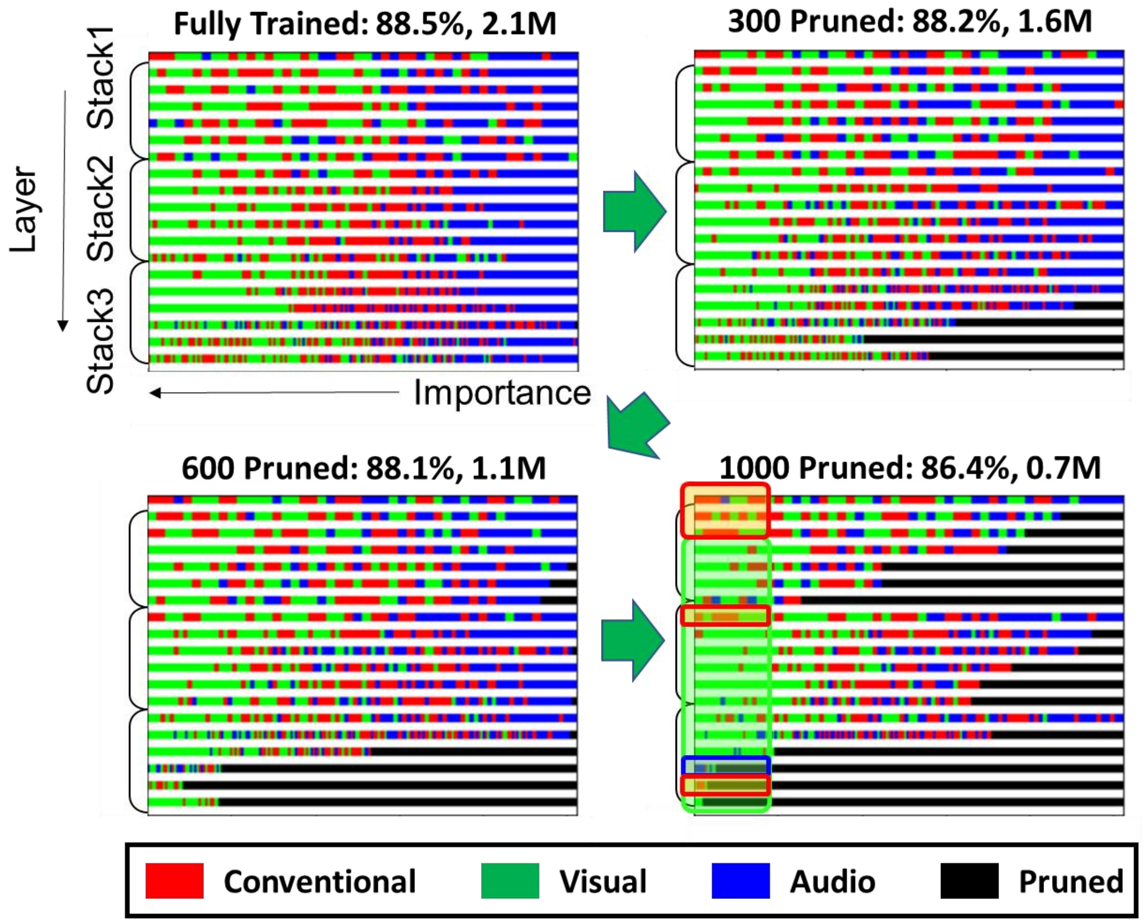


Fig.2.5.8 Importance of neurons in the network during pruning on the CiFAR-10 dataset. The network layers correspond to strips in the figure. In each strip, the conventional (ReLU), *max*, and *coincidence* neurons are by red, green, and blue, respectively, while neurons pruned are displayed in black. Neurons are ranked by their importance metric φ from the left (high φ) of each strip to the right (low φ). The initial network had an α of 88.5% and had $\eta = 2.1\text{M}$ parameters. As pruning progresses, α and η both decrease. We observe that neurons in the latter layers of the last stack were pruned first, after which neurons in the latter layers of every stack were pruned. Post pruning, we observe that conventional neurons are dominant in the early layers, *max* neurons in the middle layers, and *coincidence* neurons mainly show up in the latter layers.

Based on the importance metric φ, neurons in the 3rd stacks were pruned first, after which neurons in the 1st, then the 2nd stacks were pruned. Neurons towards the latter layers of each block had a higher probability of being pruned, indicating a bottleneck structure helps summarize the features of each stack.

In terms of *neural composition*, **Fig.2.5.8** suggests that (1) conventional neurons are important in the earlier layers (2) visual neurons are important in the middle layers, and (3)

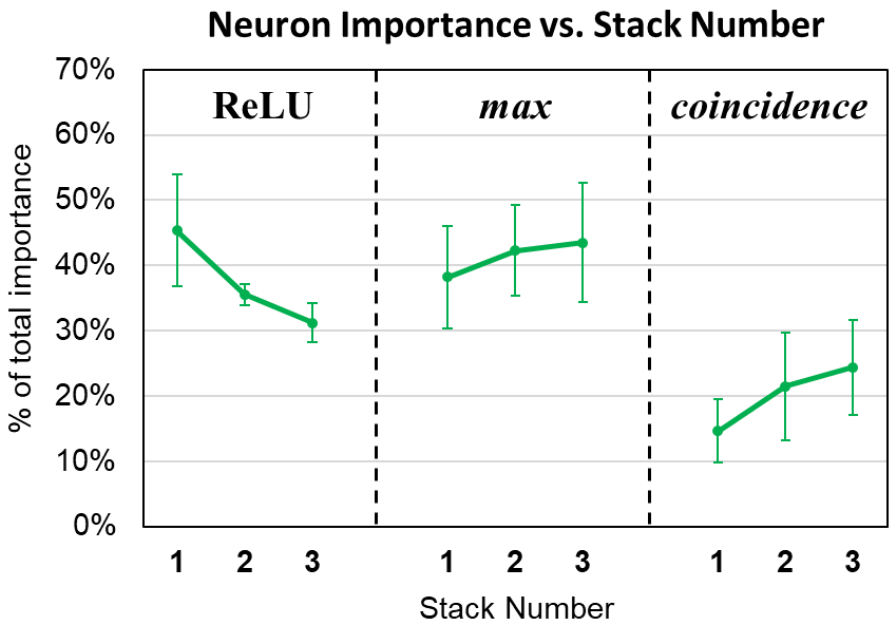


Fig.2.5.9 Pruning NAS results on the audio datasets. Most follow the same results as the CiFAR-10 dataset: (1) conventional neurons favoring early layers, visual favoring middle layers, and audio favoring latter layers; and (2) pruning increases towards the end of each block as well as towards the end of the network.

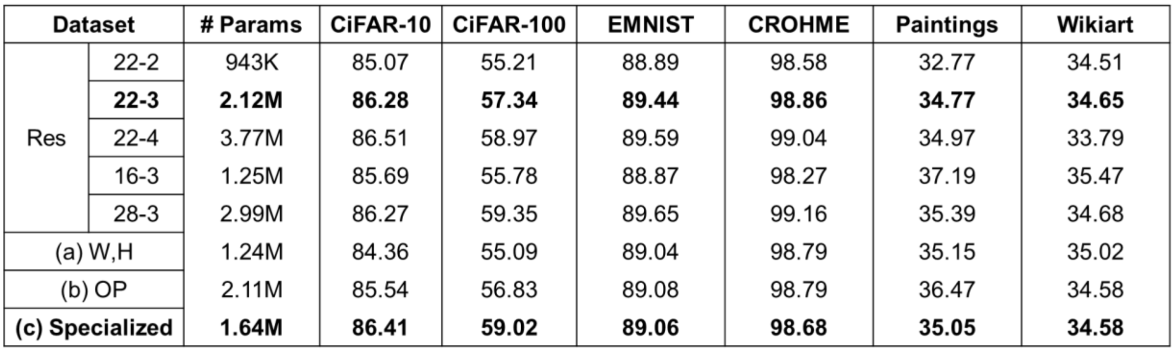
audio neurons importance rise in the latter layers. We further quantify this in **Fig.2.5.9,** which shows the sum of the φ of each neuron type divided by the sum of the φ of all neurons.

2.5d Specialized Network Design

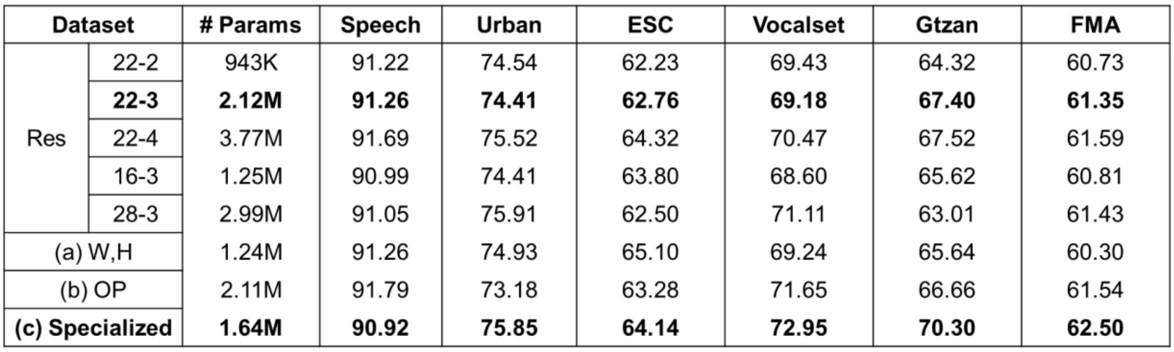
Summarizing the results in Sec.2.5a ~ Sec.2.5c, we propose design modifications to the ResNet for a more compact, high-performance architecture. Here, we take a ResNet (**Fig.2.3.8**) with $N_B = 3$ and $n_f = 48$ as an example and modify the design using the following methods:

(1) Based on the structural organization in **Fig.2.5.5**, the network should have an increased second block height and a decreased last-block height. As a result, we change the number of layers in each stack from [6,6,6] to [6,8,4].

(2) Based on the structural organization in **Fig.2.5.5**, the network should have a decreased width in the third stack. As a result, we change the number of filters in the original Resnet from [48,96,192] to [48,96,144].



Dataset		# Params	CiFAR-10	CiFAR-100	EMNIST	CROHME	Paintings	Wikiart
Res	22-2	943K	85.07	55.21	88.89	98.58	32.77	34.51
	22-3	**2.12M**	**86.28**	**57.34**	**89.44**	**98.86**	**34.77**	**34.65**
	22-4	3.77M	86.51	58.97	89.59	99.04	34.97	33.79
	16-3	1.25M	85.69	55.78	88.87	98.27	37.19	35.47
	28-3	2.99M	86.27	59.35	89.65	99.16	35.39	34.68
(a) W,H		1.24M	84.36	55.09	89.04	98.79	35.15	35.02
(b) OP		2.11M	85.54	56.83	89.08	98.79	36.47	34.58
(c) Specialized		**1.64M**	**86.41**	**59.02**	**89.06**	**98.68**	**35.05**	**34.58**



Dataset		# Params	Speech	Urban	ESC	Vocalset	Gtzan	FMA
Res	22-2	943K	91.22	74.54	62.23	69.43	64.32	60.73
	22-3	**2.12M**	**91.26**	**74.41**	**62.76**	**69.18**	**67.40**	**61.35**
	22-4	3.77M	91.69	75.52	64.32	70.47	67.52	61.59
	16-3	1.25M	90.99	74.41	63.80	68.60	65.62	60.81
	28-3	2.99M	91.05	75.91	62.50	71.11	63.01	61.43
(a) W,H		1.24M	91.26	74.93	65.10	69.24	65.64	60.30
(b) OP		2.11M	91.79	73.18	63.28	71.65	66.66	61.54
(c) Specialized		**1.64M**	**90.92**	**75.85**	**64.14**	**72.95**	**70.30**	**62.50**

Table. 2.5.2 Performance of various network architectures. For the reference ReNnet, we report α and η for networks of different sizes from $N_B = 2\sim4$ and $n_f = 32\sim64$. To confirm the importance of specialized neurons, we also report : (a) only the network structural organization changes (1) and (2), labelled as (W,H) ; (b) only the network compositional changes (3) and (4), lablled as OP. The specialized network incorporating all changes (1)~(5) (bolded, labelled as specialized), achieves improved α by an average of 2.7% over the baseline while reducing η by ~25%..

(3) Based on the structural composition in **Fig.2.5.6**, we change the operations from entirely "3x3" operations to an equal amount of "1x1", "5x5", and "7x7" in the first stack and a slightly higher "3x3", e.g., [20%, 40%, 20%, 20%].

(4) Based on the structural composition in **Fig.2.5.6**, the portion of high-K operations continuously decreases towards the end of the network. This results in a composition of in the second and third stack, the operations are changed to [20%, 50%, 20%,10%] and [30%, 70%, 0%, 0%] , respectively.

(5) Based on the neural composition in **Fig.2.5.7**, we use a composition of [45%, 40%, 15%], [35%,40%,20%], and [30%,45%,25%] of conventional, max, and coincidence neurons for 1st, 2nd, and 3rd stacks respectively.

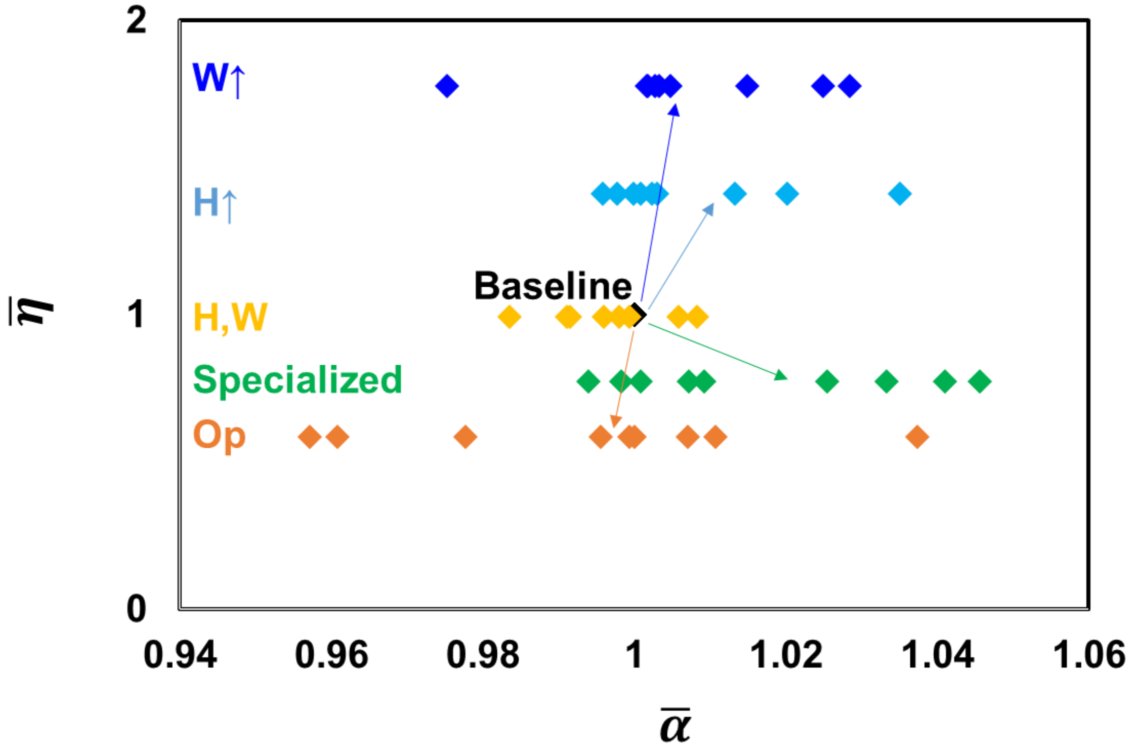


Fig.2.5.10 Scatter plot of the normalized performance of various network architectures. The reference ResNet ($N_B = 3$ and $n_f = 48$) has $\bar{\alpha} = \bar{\eta} = 1$. Larger ResNets, ($N_B = 3$ and $n_f = 64$, labelled W↑, and $N_B = 4$ and $n_f = 48$, labelled H↑) are also displayed. Network incorporating only the network structural organization changes (1) and (2) is labelled as (W,H) ; network incorporating only the network compositional changes (3) and (4) is labelled as OP. The specialized network incorporating all changes (1) ~ (5) is labelled as specialized. Overall, the specialized network achieves improved α over the baseline (also bolded) on an average of 2.7% while reducing η by ~25% and even has higher average $\bar{\alpha}$ than W↑ and H↑.

The performance of different designs are reported in **Table 2.5.2** and **Fig.2.5.11**. We show the original ResNet architecture with $N_B = 3$ and $n_f = 48$ as the baseline, as well smaller Resnets with reduced blocks per stack $N_B = 2$ and $n_f = 48$ and reduced neurons $N_B = 3$ and $n_f = 32$; larger Resnets with increased blocks per stack $N_B = 4$ and $n_f = 48$ and increased neurons $N_B = 3$ and $n_f = 64$. To verify the importance of the specialized neurons, we also show the structural organization modifications ((1) and (2) applied only, denoted as (W,H)), the structural composition modifications only ((3) and (4), denoted as OP), and finally the full specialized network ((1)-(5) all applied). Without the specialized neurons, neither the change in the structural organization nor composition can outperform the baseline. On the other hand, once the specialized neurons are employed, we observe an improved average $\bar{\alpha}$ by 2.7% as well as a reduction in $\bar{\eta}$ by 25%. In many cases, the specialized network even outperforms networks with larger depth and width with ~2x the number of parameters. These results show the importance of specialized neurons in highly efficient signal processing.

CHAPTER 3

Variation-Tolerant Memory Design

3.1 Motivation

The ever-increasing demand for memory performance has given rise to a variety of new memristive technologies with orders of magnitude improvement in programming time and energy compared to the commercially available Flash. These devices behave as electrically programmable resistors, in which the device resistance changes based on the voltage or current bias. Different resistance states correspond to different memory values; for example, a high resistance state corresponds to a digital 1, and low resistance to a digital 0. Some memristor devices also support multiple resistance states, allowing storage of multi-bit per cell. Memristive devices are also a key enabler of physical matrix-multiplication architectures.

Currently, the most popular memristor technologies include the magneto-resistive random access memory (MRAM) [80], [81], [166], the Resistive RAM (ReRAM)[167]–[169], and the Phase Change RAM (PCRAM) [37], [170], [171]. The MRAM device has a typical magnetic tunneling junction (MTJ) structure, which composes two magnetic layers separated by a tunneling barrier as shown in **Fig.3.2.1**. One magnetic layer (free layer) serves as storage, and the other (fixed layer) as a reference. Data is stored as the relative magnetic orientation of the free layer to the fixed layer: if the two layers have the same magnetic orientation, the MTJ is in its parallel (P), low resistive state (LRS); and if they have opposite orientation, the device is in its anti-parallel (AP), high resistive state (HRS). MRAMs can be programmed through a variety of spin-based phenomena, such as Spin-Transfer Torque (STT), Spin-Orbit Torque (SOT), and Voltage-Controlled Magnetic Anisotropy (VCMA) effects. They have high endurance, speed, and energy efficiency; but suffer from low distinguishability (e.g. resistance

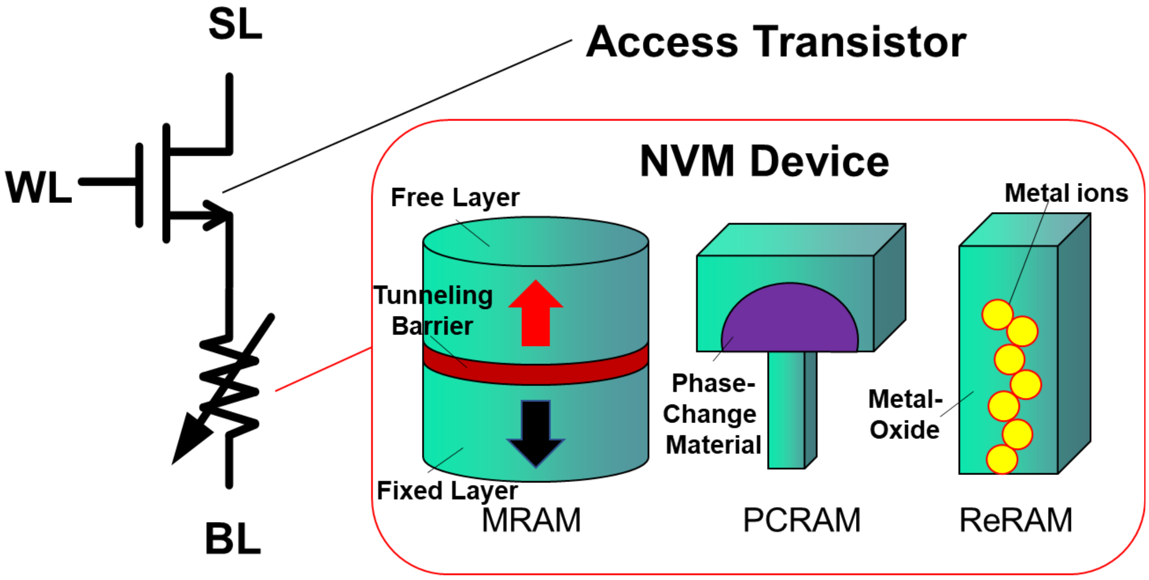


Fig.3.2.1 Structure of a 1T-1R memory cell, composing an access transistor and a memristive nonvolatile memory (NVM) device. There are various memristor technologies, such as MRAM, PCRAM, and ReRAM. A typical MRAM composes a magnetic layer-tunneling barrier-magnetic layer stack; a typical PCRAM composes a phase change material; and a typical ReRAM composes an insulating metal-oxide with a conducting metal path. The memristor is accessed by enabling the WL signal, which connects the memristor to the BL and SL.

ratio) between the two resistance states compared with other resistive memories. Thus, main application of MRAM is as a high-density, high-performance, nonvolatile embedded memory.

The Resistive RAM (ReRAM) device in general has a metal-metal oxide structure, and stores data by forming a conductive filament between the device’s two electrodes, as shown in **Fig.3.2.1**. The resistance of the device is low if a conductive path is formed, and high otherwise. ReRAMs have the advantage of high distinguishability, multi-resistance states, and easiness to fabricate given their simple structure. However, they suffer from limited endurance and large variability due to the damaging, stochastic nature of ionic filament formation. Currently, ReRAMs find their main application in storage and neuromorphic/computing-in-memory designs where write is not frequent.

The Phase-Change RAM (PCRAM) device composes of a phase-changing material and a heating element, as shown in Fig.**3.2.1**. It stores data as the crystalline structure of the phase-changing material, which displays a low resistance when it is crystalline (as it allows electrons to easily pass through); and a high resistance when it is amorphous. This phase change is achieved through different heating-cooling processes: rapid heating and cooling lead to an amorphous phase, while gradual heating/cooling results in a crystalline phase. PCRAM often has a high resistance ratio as ReRAM and the technology is production ready. It is believed to be the technology behind Intel and Micron’s 3D XPoint [172], currently available as an intermediate high-speed buffer between SSDs and the main memory. However, the heat-based write process means disturbance and gradual shifts in the device state due to thermal induced diffusion and migration.

All memristive devices suffer from variation issues. In addition to that in device dimensions (width, length, height, shape, etc) and process nonidealities (re-deposition, smoothness, sidewall impact, etc), each technology has critical factors that impact its characteristics. MRAM resistance is exponential to small changes in its barrier thickness [173]; phase change to the dimensions in the heater [174], and ReRAM from traps in the device [175]. Variation issues worsen as device dimensions scale down since the same offset contributes to a higher percentage difference. At the same time, increasing the capacity of a memory array also causes the number of tail bits (the number of memory cells at the edge of distribution) to increase. Due to variation, a given read setting will experience reduced margins and read failure while fixed write conditions could lead to wasted energy, accumulative over-programming, and write failures from resistance shifts, endurance, and stuck-at faults [176]. It is thus evident that memory variations result in a critical challenge in AI accelerators.

Here, we propose two methods to overcome the impact of variations: (1) the 2D calibration scheme, where a calibration grid is created to cancel fabrication and critical-path induced variations; and (2) the dual data line scheme, which mitigates the read challenges caused by the variations through enhancing the read signal.

3.2 Background

3.2a Memory Array Architecture

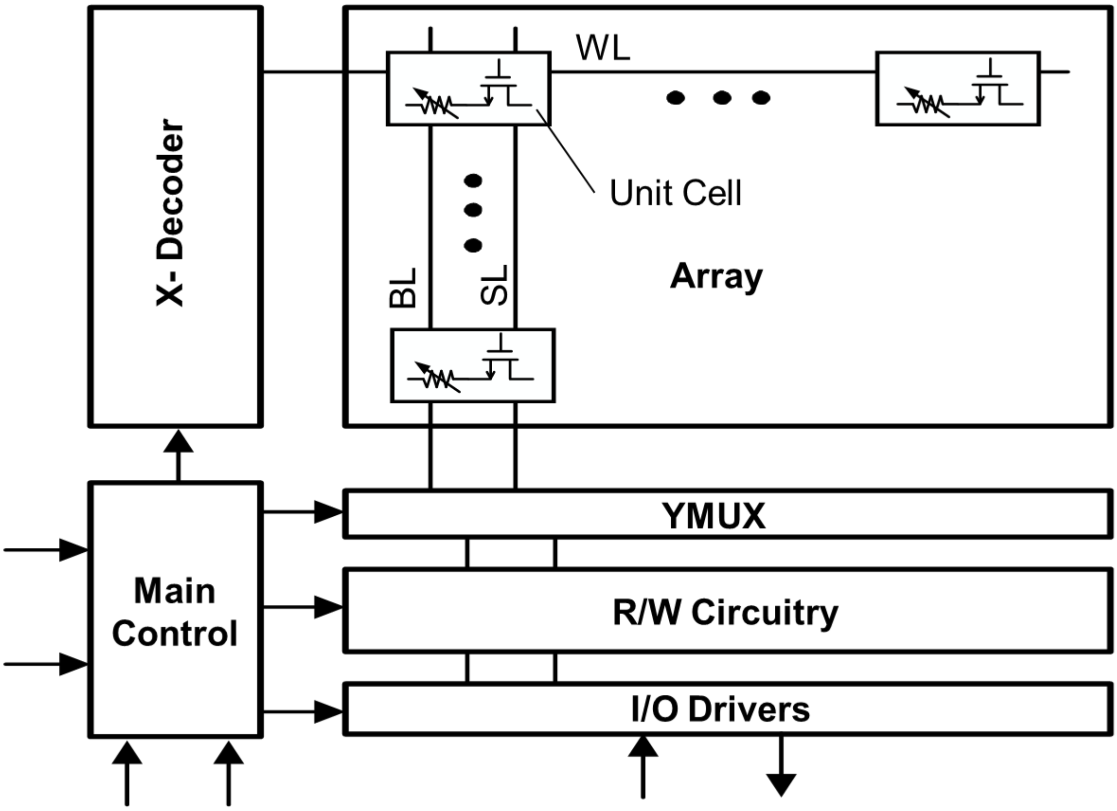


Fig.3.2.2 Structure of a memory macro, composing the main controller, X-decoder, Y-multiplexer (YMUX), an array of memory cells, read and write (R/W) circuitry, and the input/output (I/O) drivers. The main controller generates control signals for each block. The X-decoder controls the row-wise WLs, and the YMUX connects the column-wise BL and SLs to the R/W circuitry. The R/W circuitry drives the write conditions on the BL/SLs during the write operation and detects device states during the read operation. The IO /Drivers interface with the external circuitry.

The structure of a memristor memory cell is shown in **Fig.3.2.1**, comprising an access transistor and a memristive device (1T-1R cell). One electrode of the memristive device is connected to the drain of the access transistor, and the other electrode to the bit line (BL). The gate and source of the transistor are connected to the word line (WL) and the source line (SL), respectively. The cell is selected by applying a voltage to WL, which turns on the access transistor, connecting the two terminals of the device to the BL and SL.

The general structure of a memory macro is shown in **Fig.3.2.2**, which includes the main controller, an x- decoder (XDEC), a y-multiplexer (YMUX), Read/Write circuitry (RWD), and

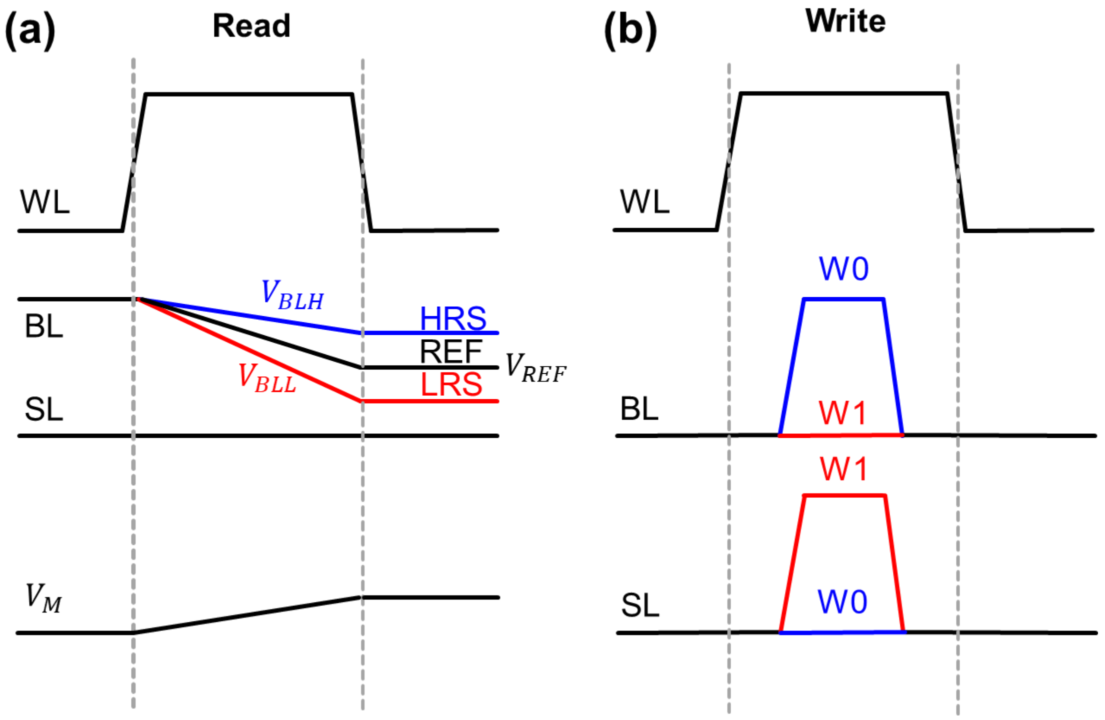


Fig.3.2.3 Read and write waveforms of a memory array. (a) During read, the BL is pre-charged high and SL grounded. When the WL is turned on, the selected cell discharges the BL rapidly if the memristor has low resistance state (red) and slowly if the cell has high resistance (blue). The reference is set mid-point between the two voltage levels. (b) During write, the BL and SL signals are driven to provide write conditions on the selected memory cell based on the input data. For example, the device could be written to a low resistance state (W0) by setting BL=VDD and SL=0 (blue); and to a high resistance state (W1) by setting BL=0 and SL=VDD (red).

an input/output interface (IO). The main controller decodes input commands such as "Read", "Write", and "Sleep", to generate the individual control signals of each of the other blocks such as "Sense Amplifier Enable", "Write Driver Enable". The x-direction decoder selects the accessed row based on the address and generates the WL signals applied to the array. Likewise, the y-direction multiplexer selects accessed column based on the address and connects the selected BLs and SLs to the R/W circuitry. The R/W circuitry composes the circuitry to read and write the memory cells. This includes sense amplifiers that are used to detect BL and SL signals, thereby generating the output data during read; as well as write drivers that drives BL and SL to the suitable biases to overwrite the memory cells during write.

The waveforms of a voltage-mode read operation are shown in **Fig.3.2.3a.** During the pre-charge phase, the BL voltage V_{BL} is initially pre-charged to the read voltage (V_{READ}) and the

SL/ WL are grounded. Afterward, the selected WL is asserted, turning on a discharge path from the floating BL through the memory cell to the grounded SL. If the selected cell is in LRS, a large discharge current results in a rapid dropping of the BL voltage (V_{BLL}). In contrast, an HRS has a relatively small current and thus a slow decrease in the BL voltage (V_{BLH}).

As V_{BL} develops, the difference between V_{BLL} and V_{BLH} increases. When this difference is sufficient, a sense amplifier compares V_{BL} to a reference voltage (V_{REF}) to determine the output data. If V_{BL} is larger than V_{REF}, the output data is determined as 1, and 0 otherwise. The reference is often chosen to be (V_{BLL} +V_{BLH})/2 so that the maximum sensing margin ($V_M = V_{BL} - V_{REF}$) can be achieved for both states. The V_{BL} development time needs to ensure that V_M can cover the offset of the sense amplifiers for a successful read operation. Normally, V_M is chosen to be at least 50mV or 100mV.

Fig.3.2.3b shows the waveforms during a write operation. During a write operation, the WL of the selected row is first asserted by the x-decoder circuit. Subsequently, the BLs and SLs are driven by the write drivers to deliver the write conditions (e.g. write voltage, time, and current) to the selected cell based on the input data.

3.2b Variation Sources

Variation caused by Fabrication

Nonidealities during fabrication often lead to a spatial distribution of device characteristics along the wafer. These spatial distributions form patterns related to specific processing steps, which include [177]–[179]:

(1) Ring patterns **(Fig.3.2.4a)**, where fabrication characteristics are proportional to its radial distance from the center of the wafer. These patterns are often caused by deposition, chemical-mechanical polishing (CMP), and their associated corrections.

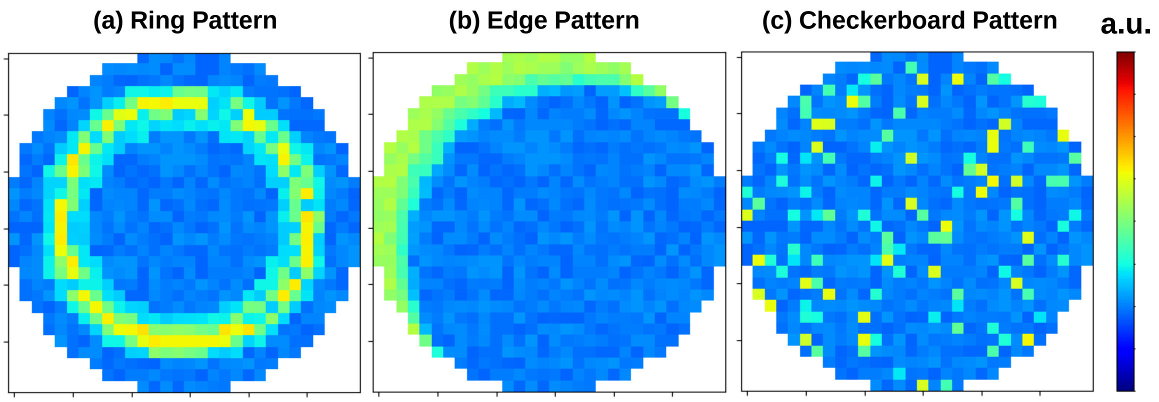


Fig.3.2.4 Common wafer variation patterns, (a) ring patterns caused by deposition and chemical-mechanical polishing, (b) Edge patterns associated with annealing, and (c) checkerboard patterns caused by misalignment. Blue colors indicate a characteristic (such as thickness of a material) shift towards lower values, while red colors indicate a shift towards higher values [177-179].

(2) Edge patterns (**Fig.3.2.4b)**, where characteristics shift towards the edge of the wafer. These patterns are often associated with batch cleaning or abnormal temperatures during annealing.

(3) Checkerboard patterns (**Fig.3.2.4c)**, where characteristics alternate between peaks. This is often caused by mask misalignment during lithography or pinned defects.

(4) Random variations, caused by stochastic processes, fluctuations, defects, and particles during fabrication (not shown).

Variation caused by Critical Path

In the conventional layout of a memory macro (**Fig.3.2.5a)**, word line (WL) drivers are located adjacent to the array, with the main control sitting below it. The read and write (R/W) circuitry is situated on the adjacent side of the array, perpendicular to the WL drivers. The delay of the critical path when accessing a memory cell consists of (a) the path from the main control to the selected WL driver, (b) the path from the selected WL driver to the addressed cell, and (c) the path from the addressed cell to the R/W circuitry.

The delay from the main controller to the selected WL driver is proportional to the access row address. WL drivers closer to the main control will receive a faster, sharper pulse while

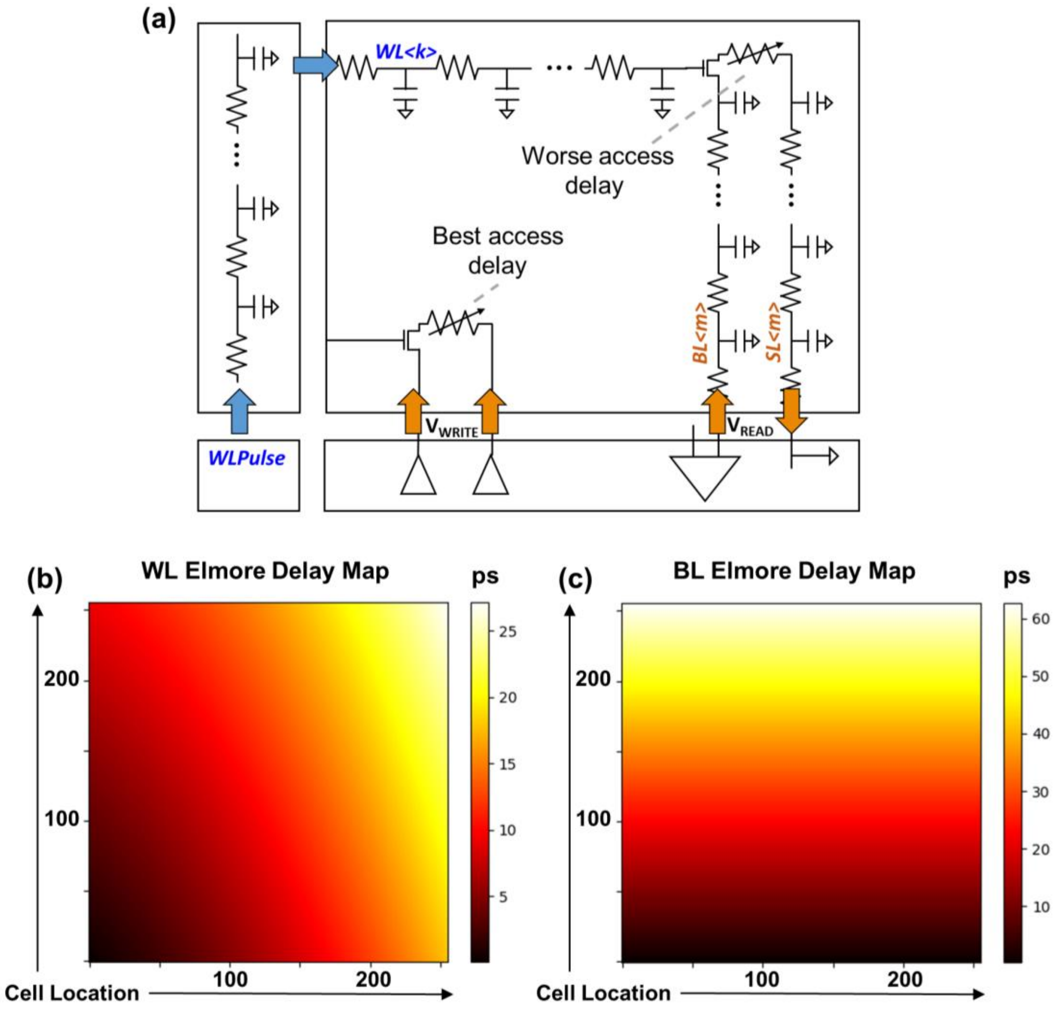


Fig.3.2.5 Delay variations caused by critical path. (a) the memory access path during read and write operations. The WL pulse (blue) goes through the decoder and the driver before arriving at the cell. Cells closer to bottom left has lower delay. The BL access is dependent of the distance to the drivers, where cells closer to the bottom of the array has faster delay. (b) heatmap of the WL access path, and (c) heatmap of the BL access path

drivers further from the controls will receive pulses that experience RC low-pass filtering. The delay from the WL driver to the addressed cell is proportional to the accessed column, as the WL pulse arriving at the cell is affected by its distance to the WL drivers. Due to the contributions from these two delays, the memory cell furthest from the main control (top right), experiences the longest delay for the WL signal. A heat map of the WL Elmore delay (a RC delay model) is shown in **Fig.3.2.5b**.

The delay from the addressed cell to the R/W circuitry is proportional to the accessed row: once the WL of the selected cell is enabled, the BLs are driven depending on the operation received. For a read operation, the BL discharge speed through the memory cell is proportional to the total resistance-capacitance path from the read circuitry to the ground. As a result, the read signal experiences additional delay when accessing cells on the top row. A heat map of the BL Elmore delay is shown in **Fig.3.2.5c** [180]. During a write operation, the write conditions received at the target cell are also proportional to the distance from the write circuitry to the cell. Write conditions received at cells furthest from the drivers experiences the most amount of degradation.

Of course, the specific design and layout of the memory macro will also contribute to other critical-path delay components.

3.3 The 2D Calibration Scheme

Calibration is often used to compensate for different variations. In practice, slower chips can be compensated by allowing longer write times or higher write voltages. The calibration of individual sense amplifiers (ex. offsets, references) and write drivers (ex. driving current) provides a finer granularity and reduces local CMOS variation to a certain extent [181], [182]. Configurable replicas can be used to track the global variation, column capacitances, and control signal timing [183]–[185]. In a tangential approach, parallel drivers and shortcut routing reduce the parasitic effects along long buses [186]. Nevertheless, none of these methods fully address variations on the critical path and none address chip-level gradients. On the other hand, the proposed 2-dimensional (2D) calibration scheme can successfully address both variation sources by interjoining row and column level calibrations to form a correction grid at each cross-point in the memory array (as detailed in **Sec.3.3a**). Compared to previous

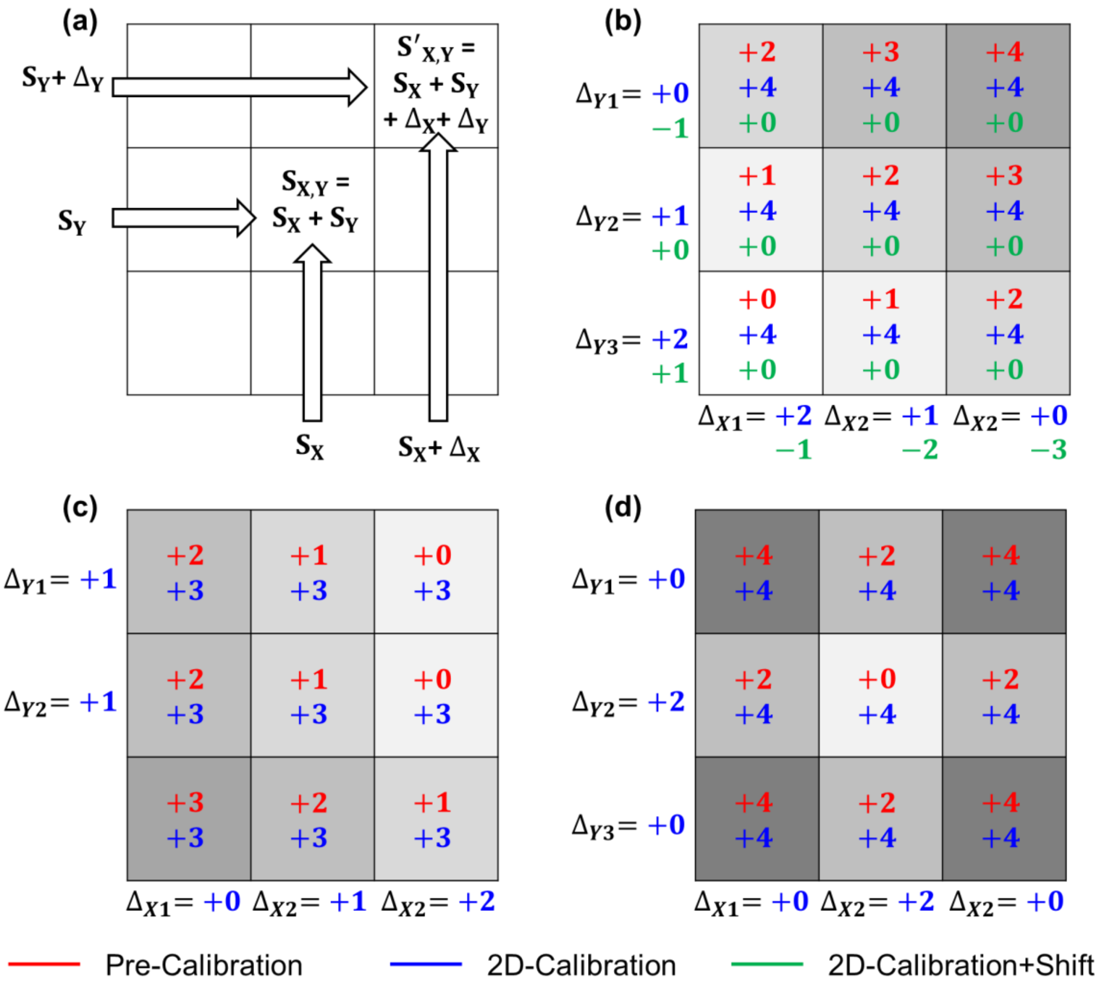


Fig.3.3.1 Illustration of the 2D calibration scheme. (a) the 2D calibration scheme is based on a design in which a crosspoint signal is the sum of the row and column signals. (b) Calibration of the WL delay map where the pre-calibration delays, displayed in red, range between 0 to 4 time units (such as nanoseconds). The 2D calibration values are displayed adjacent to the array, and post-calibration delays are displayed in each cell in blue. It can be observed that after calibration, the entire array is accessed with the same delay. If design allows, the calibration values can be shifted for a reduced delay in addition to uniformity, as displayed in green. (c) 2D calibration of an edge or ring pattern, with increasing values in the column and row directions. (d) 2D calibration of a checkerboard pattern, with alternating values between rows and columns.

calibration schemes, the post-calibration variability can be reduced by up to 99% in common variation patterns.

3.3a 2D Calibration Scheme

Our 2D calibration scheme takes advantage of a design in which the signal at a crosspoint ($S_{X,Y}$) is the sum or difference of the corresponding row signal (S_Y) and column signal (S_X), i.e. $S_{X,Y} = S_X \pm S_Y$. In this design, the calibration of the rows ($S'_Y = S_Y + \Delta_Y$) and columns ($S'_X =$

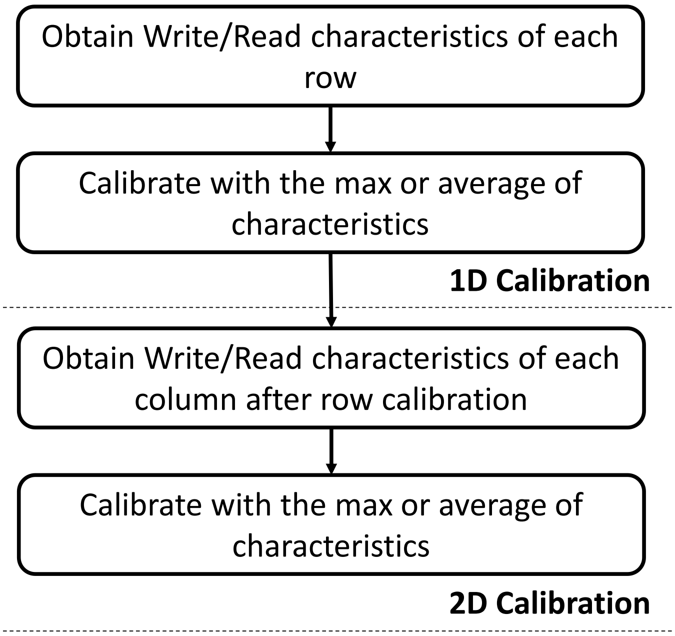

Obtain Write/Read characteristics of each row
Calibrate with the max or average of characteristics
1D Calibration
Obtain Write/Read characteristics of each column after row calibration
Calibrate with the max or average of characteristics
2D Calibration

Fig.3.3.2 Algorithm to find calibration values. 1D calibration extracts read and write characteristics of each row and computes the calibration value from its distribution, usually by finding the max range or average of the values. 2D calibration repeats this process one additional time at each row after 1D calibration.

$S_X + \Delta_X$) composes a 2-dimensional grid in which the signal at each crosspoint is corrected by the joint contribution of the row and column calibration signals, i.e. $S'_{X,Y} = S_X + S_Y + \Delta_X + \Delta_Y$. This is illustrated in **Fig.3.3.1a**.

Such a calibration grid can cancel a large diversity of spatial patterns effectively. In **Fig.3.3.1b**, we show an example of canceling a WL Elmore delay pattern with variation between 0 time units (which could be nanoseconds) for the fastest cell on the bottom left to 4 units for the slowest cell on the top right. WLs closer to the main control are calibrated with increasing delay (displayed in blue), from a +0 time unit delay at the furthest WL to an additional +2 time unit delay on the closest WL. Similarly, BLs closest to the main control are delayed by +2 time units and the furthest by +0 time units. The post-calibration results show a uniform delay of +4 time units across the entire array, demonstrating the effectiveness of the 2D calibration scheme. If the design allows calibrating using negative values (such as designs in which WL pulses can arrive early), the calibration values can be linearly shifted to improve the performance of the array, as displayed in green in **Fig.3.3.1b**. Rows are calibrated with values from -1 to +1 time units and columns from -1 to -3 time units, allowing all cells in the array to be accessed with 0 delay.

Other common fabrication patterns of delay variation can be calibrated as follows: Edge and ring-like patterns can be calibrated with increasing/decreasing values in the direction of the gradient (**Fig.3.3.1c)**. Checkerboard patterns can be calibrated via interleaving high and low values in both the column and row directions (**Fig.3.3.1d**). A combination of patterns can be achieved via a superposition of the calibration values corresponding to each pattern.

The algorithm to find calibration values for the 2D calibration scheme is shown in **Fig.3.3.2**. It follows the same steps as conventional calibration algorithms: (1) detecting characteristics of cells, then (2) calibrating via the average or max amount of variability. The global calibration applies this at the macro level while the 1D calibration does this at the column level. The 2D calibration adds an additional row-level calibration step after the column-wise calibration, e.g., after the column-level calibration, the post-calibration characteristics of each row are obtained to compute the row-level calibration values. This overhead is necessary only once during each calibration, usually during chip testing.

3.3b Circuit Implementation

Traditional non-overlapping control signals do not necessarily allow a crosspoint signal to be the sum or difference of the row and column signals. In this section, we address the circuit implementation of the 2D calibration scheme. Specifically, we demonstrate the modifications to traditional designs necessary to enable 2D calibration of delay.

The modification for write operation with 2D calibration is shown in **Fig.3.3.3a**. In the conventional write operation, the row WL and column BL signals are nonoverlapping. The write time of the device (t_w) is determined purely by the BL pulse timing, e.g. the difference between the BL pulse start time (t_{BLS}) and the BL pulse end time (t_{BLE}). Adjusting the WL

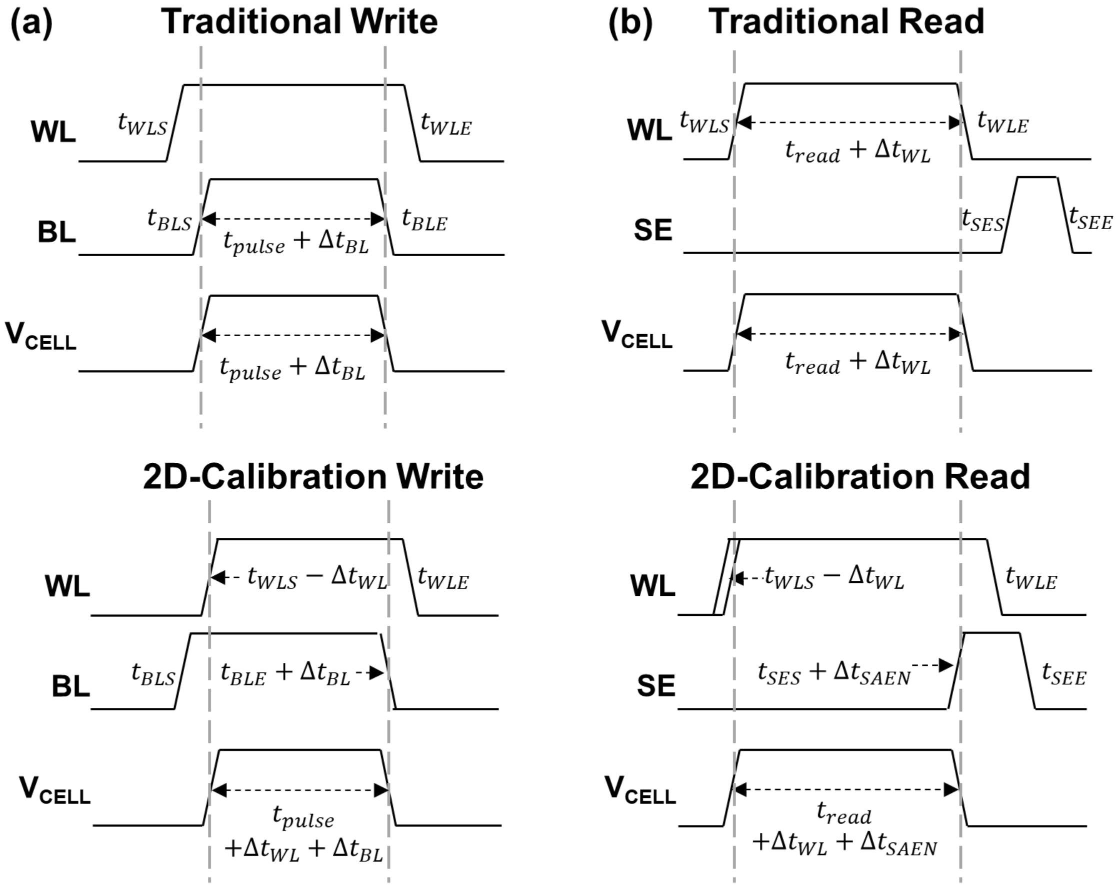

(a)
Traditional Write
WL
t_{WLS}
t_{WLE}
BL
t_{BLS}
$t_{pulse} + \Delta t_{BL}$
t_{BLE}
V_{CELL}
$t_{pulse} + \Delta t_{BL}$
(b)
Traditional Read
WL
t_{WLS}
$t_{read} + \Delta t_{WL}$
t_{WLE}
SE
t_{SES}
t_{SEE}
V_{CELL}
$t_{read} + \Delta t_{WL}$
2D-Calibration Write
WL
$t_{WLS} - \Delta t_{WL}$
t_{WLE}
BL
t_{BLS}
$t_{BLE} + \Delta t_{BL}$
V_{CELL}
t_{pulse}
$+\Delta t_{WL} + \Delta t_{BL}$
2D-Calibration Read
WL
$t_{WLS} - \Delta t_{WL}$
t_{WLE}
SE
$t_{SES} + \Delta t_{SAEN}$
t_{SEE}
V_{CELL}
t_{read}
$+\Delta t_{WL} + \Delta t_{SAEN}$

Fig.3.3.3 Read and write design modifications for the 2D calibration. (a) The conventional write waveforms (top) and the 2D calibration write waveforms (bottom). The write pulse for traditional write is enabled entirely by the BL. As a result, the voltage pulse received at the cell, V_{CELL}, is solely determined by the BL pulse. On the other hand, the write pulse of the 2D calibration scheme is enabled (started) by the WL and disabled (ended) by the BL. This allows the write pulse received at the cell to have the joint contribution of the BL and the WL calibrations. (b) The conventional read waveforms (top) and the 2D calibration read waveforms (bottom). The read time for the traditional read is enabled entirely by the WL. On the other hand, the read time of the 2D calibration scheme is enabled by the WL and disabled by the Sense Enable (SE) signal. This allows the read time of a cell to have the joint contribution of the WL and SE calibrations.

timing does not affect the write time of the device. Therefore, the calibration of the write pulse can only be done by a change in the BL pulse by Δt_{BL}. For 2D calibration, we intersect the BL and WL signals such that the write time is enabled by WL and ends through BL, e.g., the write time is the difference between the WL start time t_{WLS} and the BL end time t_{BLE}. In this design, a change in the WL timing t_{WLS} by Δt_{WL} and a change in the BL timing t_{BLE} by Δt_{BL} will result in the write pulse being changed by (Δt_{WL} + Δt_{BL}). The values for each Δt_{WL} and Δt_{BL} can be obtained using the algorithm in **Fig.3.3.2**.

The modification of a read operation with 2D calibration is shown in **Fig.3.3.3b**. In the conventional read operation, the row WL signal and the column sense amplifier enable (SE) signals are nonoverlapping. Therefore, the read development time is determined solely by the WL timing. To enable 2D calibration read, the read operation is enabled by the WL start time t_{WLS}, and sampled at the SE start time t_{SES}. In this setting, a change in t_{WLS} by Δt_{WL} and a change in t_{SES} by Δt_{SE} will result in the read time being changed by $\Delta t_{WL} + \Delta t_{SE}$.

These circuit modifications can be easily extended for voltage and current calibrations in a similar sense. The drawback is that the overhead for generating granulated voltage and current steps is usually much higher than generating temporal steps. Another challenge is that the voltage on each memory device is directly affected by the BL and SL voltage, but indirectly affected by the WL voltage. A nonlinear mapping or a horizontal-SL-vertical-BL array structure may be necessary for 2D calibration of voltage or current.

3.3c Simulation Results

We demonstrate the performance of the 2D calibration scheme on a 256x256 1T1R array in 28nm technology. Parasitic capacitances and resistances are extracted via parasitic extraction (PEX) from the center cell of a 3x3 layout with WLs running on horizontal Metal3 and BLs/SLs running on the vertical Metal 2. For fabrication variations, we take the highest-varying regions in the ring, edge, and checkerboard patterns, and allow device resistance variation up to 10% across the array [187]. The gradient direction is selected such that the fabrication and access path contributions sum to create the worst-case variation condition, as shown in **Fig.3.3.4a**. For the global, 1D, and 2D calibration schemes, we plot the post-calibration Elmore delay in **Fig.3.3.4b-d** and report the post-calibration statistics (mean value of the variation var_{mean}, maximum value of the variation var_{max}, and standard deviation of the variation var_{std}) of each calibration method in the corresponding figures.

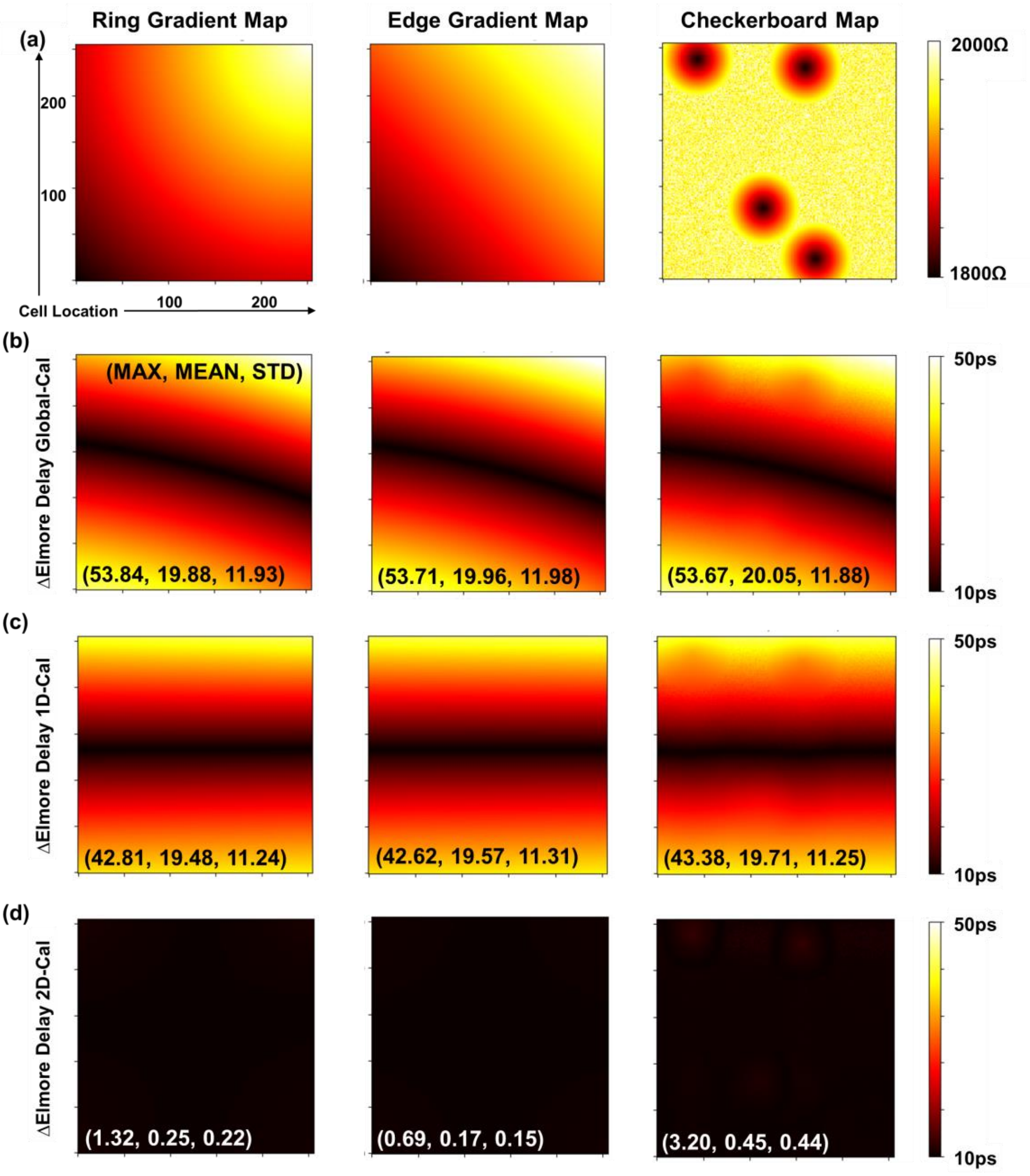


Fig.3.3.4 Performance of global, 1D, and 2D calibrations on various wafer patterns. (a) The three common wafer patterns: ring, edge, and checkerboard patterns. A resistance variation up to 10% is allowed across the array. (b) Post-calibration delay distribution of global calibration. Legend numbers show the max, mean, and standard deviation of the delay. Contributions of the WL access path and the wafer pattern remain visible. (c) Post-calibration delay of the 1D calibration. Most inter-column variations are cancelled, while row-wise variations remain. (d) Post-calibration delay of the 2D calibration, with >90% improvement over 1D calibration (a) and global calibration (b) across all patterns.

While global calibration can overcome wafer-level variations, it fails to address variations across the array. An obvious contribution of the WL and BL access path with additional shifts from fabrication gradients can be observed in all three patterns. The 1D calibration achieves a

~20% reduction in var_{max} compared to global calibration across different wafer patterns, but only a small improvement in var_{mean}. The 1D calibration can greatly reduce column-wise variation, but row-wise variations remain. The 2D calibration successfully cancels the majority of variation patterns and significantly reduces the variation across the array. Compared to the global calibration, the 2D calibration reduces var_{max} by 98%, 99%, and 94% for the ring, edge, and checkerboard patterns, respectively, and var_{mean} by 99%, 99%, and 98%. Compared to the 1D calibration, the 2D calibration reduces the var_{max} by 97%, 98%, and 93%; and var_{mean} by 99%, 99%, and 98%.

A drawback of the 2D calibration is that it requires twice the number of calibration bits as the 1D calibration. Therefore, a comparison of the calibration efficiency, e.g., calibration performance with respect to the number of calibration bits (n_{cal}) is necessary. **Fig.3.3.5** shows the performance of each calibration scheme as a function of n_{cal}. We assume that the available calibration values are evenly distributed between the maximum and minimum delays (d_{max}, d_{min}), with a step size of $(d_{max} - d_{min})/2^{n_{cal}}$. We observe that var_{mean} of the global calibration shows a small improvement with n_{cal} as it only uses a single calibration value. We also observe that the calibration settings for the lowest var_{mean} does not necessarily correspond to that for lowest var_{max} , as var_{mean} depends on the entire distribution whilevar_{max} depends on the worst-case value. For 1D calibration, var_{mean} improves with n_{cal}, but saturates above $n_{cal} = 4$ with a post-calibration variation of 19ps. A similar effect is observed in var_{max}, where a small improvement is observed above n_{cal} =6.

On the other hand, the 2D calibration steadily improves var_{mean} and var_{max} with increasing n_{cal}. Across different values of n_{cal}, var_{mean} is reduced from 12.15ps at $n_{cal} = 1$ down to 0.25ps at $n_{cal} = 10$. Comparing 1D and 2D calibration at the same $n_{cal} = 1$, the 2D calibration reduces var_{max} by 35% and the var_{mean} by 41 %. Comparing the 1D and 2D

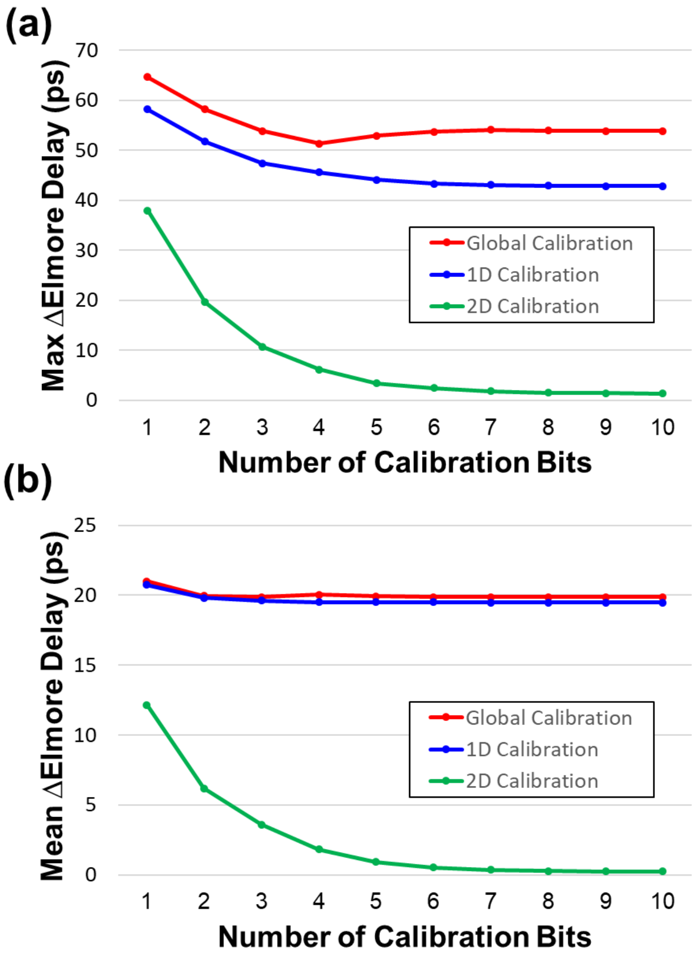


Fig.3.3.5 Performance of different calibration methods with respect to the number of calibration bits n_{cal}. (top) the maximum variation var_{max}, and (bottom) the mean variation var_{mean}. The performance of Global and 1D calibration both improve with n_{cal}, but plateaus above a certain number of bits. 2D calibration outperforms both methods and continues to decrease variation across the array with higher n_{cal}.

calibration schemes under the same overhead (i.e. 1D calibration with $n_{cal} = 2$ and 2D calibration with $n_{cal} = 1$), 2D calibration reduces the maximum variation by 27% and the average variation by 39%.

As mentioned earlier, the 2D calibration forms a calibration "grid", as compared to the 1D calibration which form column-wise "lines". This grid is more observable when there is a limited number of quantized calibration values. For example, **Fig.3.3.6** shows the distribution of post-calibration delay in the array with $n_{cal} = 4$, where the 2D calibration grid is clearly visible.

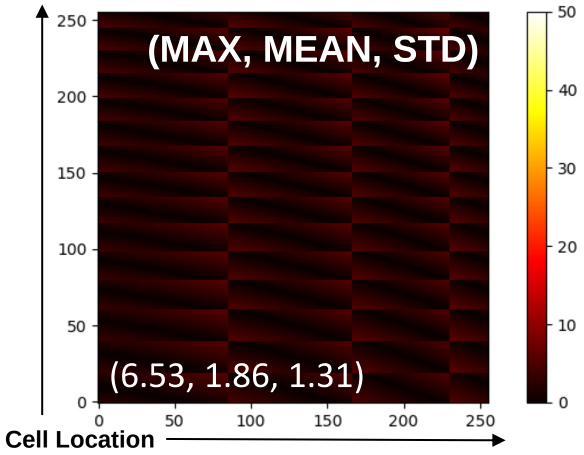


Fig.3.3.6 A post-calibration heatmap of 2D calibration with $n_{cal} = 4$. Under a limited amount of calibration values, the 2D calibration grid acting upon the original variation pattern is observable. This grid is in contrast with the 1D calibration, where column-wise calibration forms "lines" instead. The 2D calibration results are often best in the center of each grid location.

The simulated BL voltage waveforms with and without 2D calibration are shown in **Fig.3.3.7.** We display the waveforms of memory cells with the largest variation in delay during access as waveforms of all other cells will reside between them, e.g. cells at the four corners of the 256x256 array: cell[0,0], cell[0,255], cell[255,0], and cell[255,255]. The memory cells have a low resistance of 2k and high resistance of 10k, and the access delay is calibrated at a target read time of 150ps. The 2D calibration significantly improves the read performance. For example, to reach a 50mV margin, the read time can be reduced by 40% from 250ps to 150ps while for the same read time of 250ps, the margin can be improved by over 50%.

A summary of different calibration schemes is shown in **Table 3.3.I**. With $n_{cal} = 4$, the 1D calibration requires an overhead of 1.28% (e.g., 4 calibrations bits per column of 256 bits with an array efficiency of 80%) and provides a calibration improvement (characterized by the max, mean, and standard deviation in the delay variation) of ~10% compared to global calibration. The 2D calibration uses twice the amount of calibration bits as 1D calibration, resulting in a 2.56% area overhead. However, it provides a significantly larger improvement in calibration performance by between 800%~1000%.

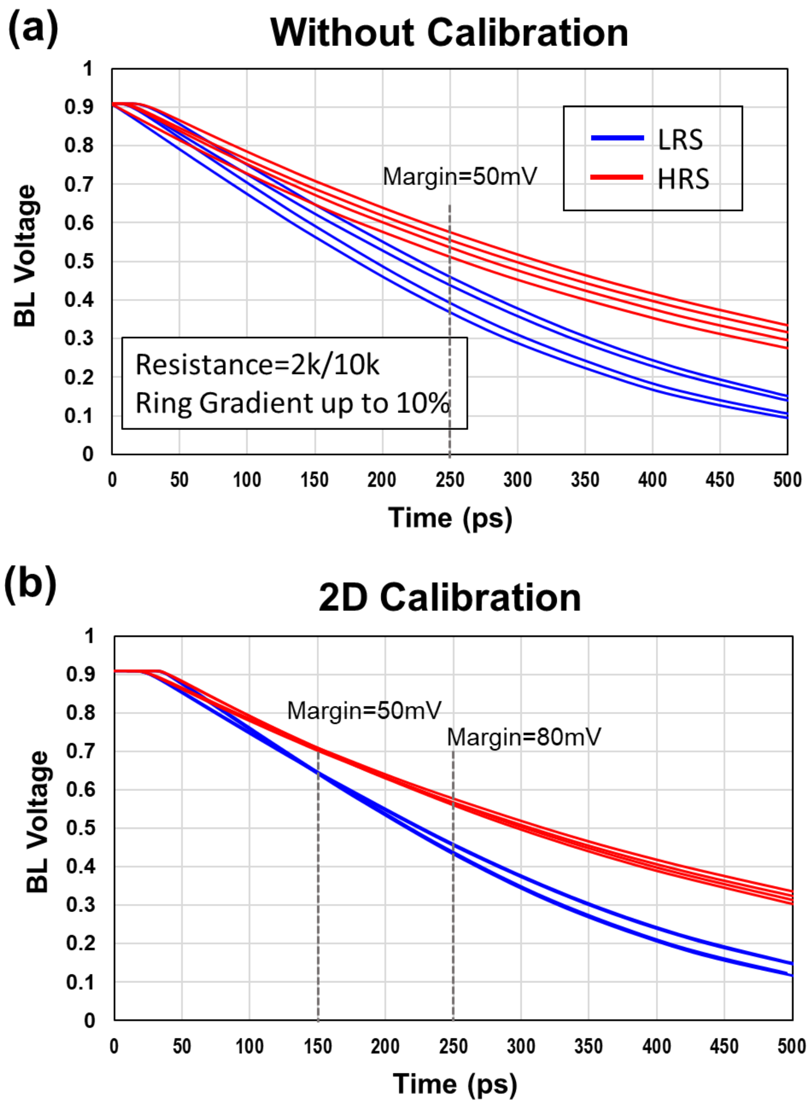


Fig.3.3.7 BL waveforms of the worst-case cells during read (a) without calibration, and (b) with 2D calibration. The 2D calibration targets the read condition of 50mV read margin. For this amount of margin, the 2D calibration achieves a reduction in read time from 250ps to 150ps. Given the same read time, the 2D calibration improves the read margin by over 50%.

Table 3.3.1. Performance and overhead of different calibration schemes

4 calibration bits 256x256 array	**Global Calibration**	**1D Calibration**	**2D Calibration**
Calibration performance*	(1,1,1)	(1.12, 1.02, 1.05)	(8.37, 11.01, 9.30)
Area Overhead	<1%	1.28%	2.56%

*performance = delay variation max^{-1}, mean^{-1}, std $^{-1}$

3.4 The Dual-Dataline Sensing Scheme

In contrast to compensating for variation, the dual-data line (DDL) sensing scheme alleviates variation-induced read challenges by improving the read signal (thereby improving the circuit tolerance to variation) instead. This is achieved by reusing read current from the bit line (BL) during read to create a differential voltage swing on the source line (SL) and combining the two to create a larger sensing signal. This increased signal provides several advantages: for the same amount of margin, the sensing time is reduced, allowing for lower energy and higher speed; while for the same sensing time, the margin is increased, improving the reliability of the read operation. We then show circuit modifications to the traditional and offset-canceling sense amplifiers to utilize the differential signals of DDL. Simulation results show that the sensing time and energy can be reduced by 47% and 48%, respectively, given the same sensing margin requirement; or under the same sensing time, the sensing margin can be increased by 86%.

3.4a Dual-Dataline Sensing Scheme

An illustration of the DDL sensing scheme is shown in **Fig.3.4.1**. Instead of grounding the SL during the read operation, the DDL scheme pre-discharges SL by connecting it to ground only during the pre-charge phase, then leaves it floating as the BL begins to develop. In this design, the current that is discharging the BL will charge the SL, resulting in a similar differential voltage swing on SL. The voltage difference between V_{BL} and V_{SL} (i.e. $V_{BL}-V_{SL}$) is applied to a sense amplifier, which sums the contribution of the two voltage swings and compares it with a reference to generate the output. The waveforms during the read operation are shown in **Fig.3.4.1b**. If the selected cell is in LRS, a large cell current causes a rapid discharge of the BL and a fast charge of the SL (V_{BL_LRS} and V_{SL_LRS}). Conversely, a cell in an

HRS state results in a small cell current, and the BL/ SL voltages discharges/ charges slowly (V_{BL_HRS} and V_{SL_HRS}).

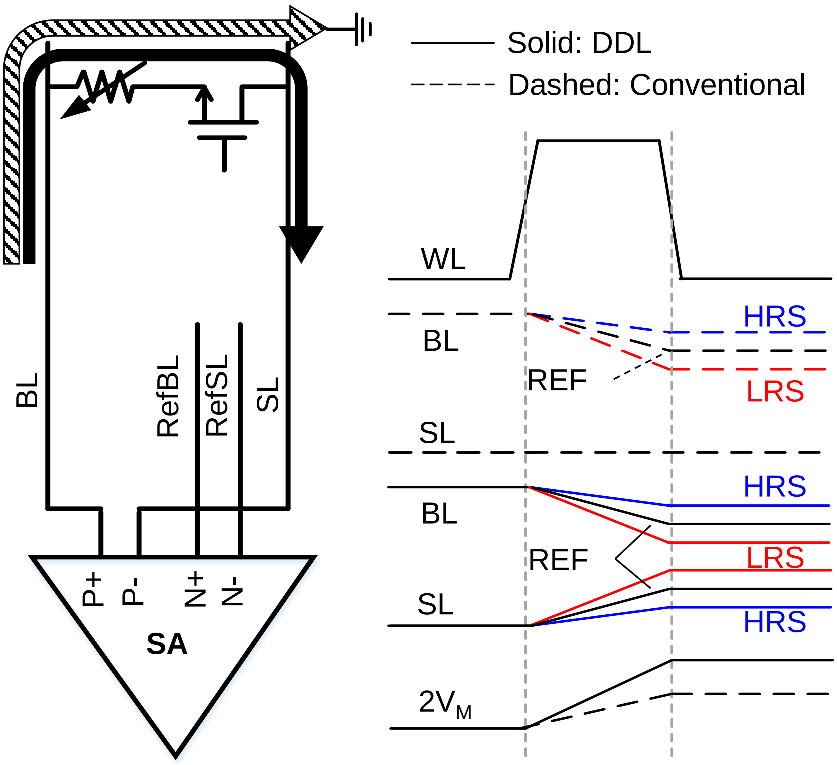


Fig.3.4.1. Illustration of the DDL scheme. (left) Read current path and (right) read waveforms. In the conventional design, the BL is pre-charged and SL grounded. During read, the floating BL discharges through the memory cell. Reading a memory cell with high resistance (HRS, blue) discharges the BL slower, while a cell with low resistance (LRS, red) would discharge the BL faster. A reference cell (REF, black) would create a reference signal with a discharge speed between HRS and LRS. The read margin (V_M) is equal to the difference between the HRS or LRS signals and the REF signal. In the DDL scheme, the BL is pre-charged and SL pre-discharged. During read, the BL discharges through the memory cell while the SL charges by the discharge current, resulting in a differential swing on the BL and SL.

The theoretical performance improvement of the DDL scheme can be obtained through passive circuit analysis. Without loss of generality, we make the following assumptions in the analysis: (1) the resistance of the memory cell selected is constant during the read operation, (2) the resistance of the access transistor is small (<0.1x) compared to that of the memristor, and (3) the BL/SL voltages do not saturate (i.e. approach a steady-state value) during the read process. These approximations are justified for a small voltage swing on the BL, which is usually desired during read.

In the conventional voltage-mode read operation, the BL voltage follows an RC discharge with the memory cell resistance and the BL capacitance:

$$V_{BL\_CONV}(t) = V_{READ} e^{\frac{-t}{R_{CELL}C_{BL}}} \quad \ldots Eq.3.4.1$$

Where V_{BL_CONV} is the BL voltage during the conventional read operation, V_{READ} iis the BL precharge voltage, R_{CELL} is the resistance of the accessed memory cell, and C_{BL} is the BL capacitance. The optimal read margin (V_{M_CONV}) is half of the BL voltage difference between reading an HRS cell and an LRS cell, i.e.

$$V_{M\_CONV}(t) = 0.5V_{READ}\left(e^{\frac{-t}{R_H C_{BL}}} - e^{\frac{-t}{R_L C_{BL}}}\right) \ldots Eq.3.4.2$$

Where R_L is the resistance of a LRS cell, R_H is the resistance of a high resistance cell, and V_{M_CONV} is the read margin. In the DDL read operation, the BL and SL voltages can be described as an RC discharge and charge towards the equilibrium charge redistribution:

$$V_{BL\_DDL}(t) = V_{READ}\left(\frac{C_X}{C_{SL}} + \frac{C_X}{C_{BL}} e^{\frac{-t}{R_{CELL}C_X}}\right) \quad \ldots Eq.3.4.3$$

$$V_{SL\_DDL}(t) = V_{READ}\frac{C_X}{C_{SL}}\left(1 - e^{\frac{-t}{R_{CELL}C_X}}\right) \quad \ldots Eq.3.4.4$$

where $C_X = C_{BL}C_{SL}/(C_{BL} + C_{SL})$ is the serial capacitance of C_{BL} and C_{SL}. When $C_{BL} \approx C_{SL}$, the optimal DDL margin V_{M_DDL}, defined as half the difference in $(V_{BL} - V_{SL})$ between reading an HRS cell and an LRS cell, is given by

$$V_{M\_DDL}(t) = 0.5V_{READ}\left(e^{\frac{-2t}{R_H C_{BL}}} - e^{\frac{-2t}{R_L C_{BL}}}\right) \quad \ldots Eq.3.4.5$$

Comparing Eq.3.4.2 and Eq.3.4.5, we find that the DDL scheme can achieve the same amount of margin as the conventional method with half the BL development time.

Eq.3.4.3 and Eq.3.4.4 show that the improvement from the DDL scheme will differ under different BL and SL loadings. If the SL has a relatively small loading compared to the BL, the voltage swing on SL will be larger than that of the BL. The improvement from the DDL scheme

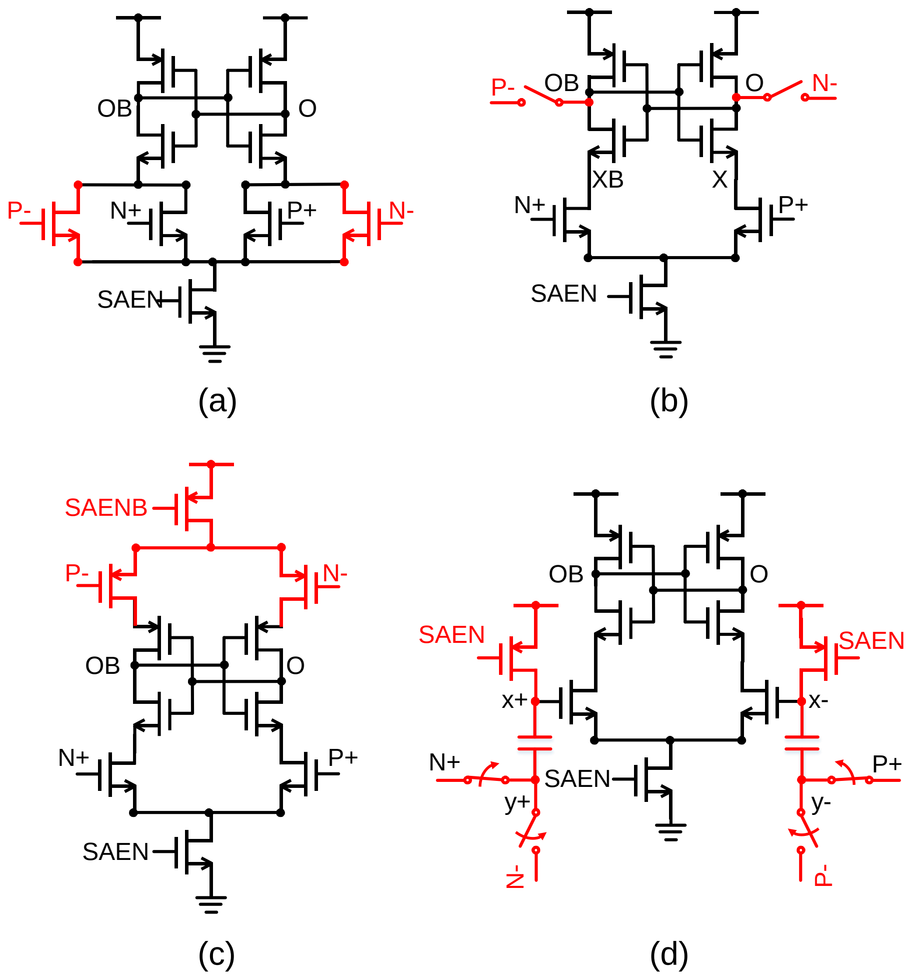


Fig.3.4.2. Sense amplifier designs to utilize the DDL scheme. The original sense amplifier is displayed in black, and the modifications in red. (a) directly adding a pair of pull-down transistors as the second pair of inputs, (b) using initial values to nodes along the sensing path as the second pair of inputs, (c) adding a pull-up evaluation path in addition to the original pull-down path, and (d) storing the inputs in capacitors and using coupling to sum the signals.

improvement increases to above 2x compared to conventional sensing. On the other hand, if SL has a relatively large loading, the swing on the SL will be smaller than that on the BL, and the improvement of the DDL will reduce to between 1x~2x. Nevertheless, the DDL scheme will always outperform the conventional scheme.

The energy consumed during a read operation is the energy necessary to pre-charge the BL voltage:

$$E_{READ} = C_{BL}V_{READ}(V_{READ} - V_{BL}) \quad \ldots Eq.3.4.6$$

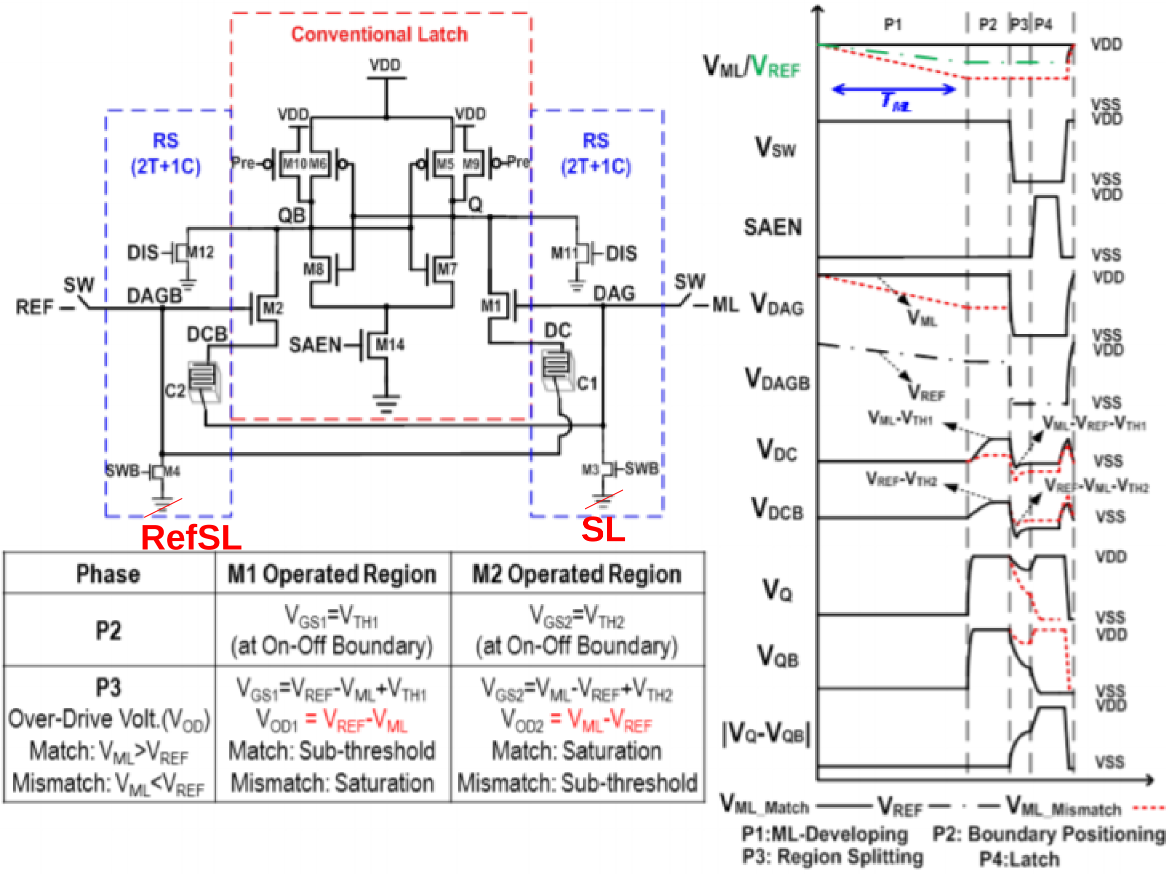
Since the DDL only requires half the BL swing for the same amount of margin, a ~50% energy reduction is achieved for the same margin. These first-order approximations agree with the simulation results in Section 3.4c.

3.4b Circuit Implementation

Traditional sense amplifiers (SA) can only compare a single signal (e.g., V_{BL}) to a single reference (e.g., V_{RefBL}). To incorporate the DDL scheme, the sense amplifier (SA) needs to be able to take in the differential signals V_{BL} and V_{SL}, and compare $(V_{BL} - V_{SL})$ with $(V_{RefBL} - V_{RefSL})$. We show several approaches to achieve this via modification of conventional SAs in **Fig.3.4.2**. The modified SAs have inputs N+, N-, P+, and P-, connected to V_{BL}, V_{SL}, V_{RefBL} and V_{RefSL}, respectively. The output of the SA is differential signals O and OB.

Fig.3.4.2a shows an implementation of the sense amplifier by directly adding an additional input pair of NMOS inputs (displayed in red). The pull-down path for node OB is driven by N+ and P-, while that of node O is driven by N- and P+. The advantage of this design is its simplicity and matching input pairs. However, as V_{BL} and V_{SL} have different DC voltage levels (the former pre-charged to V_{DD} and the latter to V_{SS}), their effects on the pull-down path are asymmetric. In particular, V_{SL} needs to exceed the threshold voltage of the NMOS input transistors to activate the input.

Fig.3.4.2b shows an implementation where the output nodes are used as the second pair of inputs. In this design, the outputs O and OB are initially connected to N- and P-. The pulldown path of O is driven by P+, while that of OB is driven by N+. When the sense amplifier is enabled, the output is the result of comparing “N- discharged by P+” and “P- discharged by N+”. Compared to the design of **Fig.3.4.2a**, this method does not add additional transistors on the sensing path and thus has a smaller impact on the original design. However, the addition of switches in the SA necessitates additional control signals and suffers from dynamic coupling



Phase	M1 Operated Region	M2 Operated Region
P2	$V_{GS1}=V_{TH1}$ (at On-Off Boundary)	$V_{GS2}=V_{TH2}$ (at On-Off Boundary)
P3 Over-Drive Volt.(V_{OD}) Match: $V_{ML}>V_{REF}$ Mismatch: $V_{ML}<V_{REF}$	$V_{GS1}=V_{REF}-V_{ML}+V_{TH1}$ $V_{OD1}=V_{REF}-V_{ML}$ Match: Sub-threshold Mismatch: Saturation	$V_{GS2}=V_{ML}-V_{REF}+V_{TH2}$ $V_{OD2}=V_{ML}-V_{REF}$ Match: Saturation Mismatch: Sub-threshold

Fig.3.4.3. The RS-SA featuring offset cancellation and on-off boundary operation (Figure adopted from [190]). The read operation composes four phases: In phase 1, the BL develops and is sampled into capacitors via nodes DAG/DAGB. In phase 2, the offset of the amplifier is sampled into the capacitors. At the same time, the comparison transistors (M1, M2) are set to operate at the on-off boundary via nodes DC/DCB. In phase 3, DAG/DAGB are coupled to VSS, causing the input pair to operate in different regions: saturation and subthreshold for the higher and lower inputs, respectively. The difference is amplified to generate the sensing result. In phase 4, the output is latched. The DDL scheme can be adopted by connecting the negative boosters to SL and RefSL instead of VSS in phase 3 and 4, as displayed in red in the schematic.

noise. In addition, each input node also has voltage-level limitations. This concept can also be extended to other nodes on the sensing path, such as nodes X and XB for inputs.

Fig.3.4.2c adds an additional pull-up path to evaluate P- and N-. The advantage of this configuration is that for the default DDL design in which V_{SL} is pre-discharged and V_{BL} pre-charged, all input transistors are fully turned on. The drawback is that the PMOS inputs may have considerably different characteristics (variation, mobility, threshold, etc.) than the NMOS input.

Fig.3.4.2d shows an adaptation of the DDL scheme by capacitively coupling the swings of N+ and P+ to that of N- and P- for summation. One pre-charge transistor, a capacitor, and two switches are added to each input of the original SA. One node of the capacitor (nodes x+ and x-) is connected to the input of the conventional sense amplifier, and the other node (node y+/y-) is switched between V_{BL} and V_{SL}. Initially, node x is precharged to V_{DD}, and node y is connected to BL. A voltage difference of (V_{DD}- V_{BL}) is stored across the capacitor. When SAEN is enabled, node x becomes floating while node y is switched to V_{SL}. Since the voltage across the capacitor stays constant, node x couples to $(V_{SL} + V_{DD} - V_{BL})$. This design has advantages including (1) V_{SL} and V_{BL} have symmetric effects on the input, (2) the working input voltage range is significantly larger, and (3) The capacitors can further utilized to cancel offsets of the SA. The disadvantage of this approach is the large area and crosstalk issues during the coupling process.

The design and optimization of the SA structure depend on the resistance characteristics of the employed device and the read voltage level. The tradeoff between the different designs in **Fig.3.4.2** can be used in the development of a new SA to fully utilize the benefits of DDL. For example, the DDL scheme can be combined with offset canceling or margin enhancing amplifiers [80], [188], [189] to achieve additional improvement in performance. One particular SA that can directly incorporate and benefit from the DDL scheme is the Region-Splitter Sense Amplifier (RS-SA) shown in **Fig.3.4.3** [190]. The RS-SA uses the mechanism of **Fig.3.4.2d**, and additionally features (1) storing the offset of the amplifier in the capacitor, and (2) operates at the on-off boundary of the pulldown transistors. The DDL scheme is incorporated by connecting the negative boosters (M3 and M4) to SL and RefSL instead of VSS. With this simple modification, the DDL scheme halves the sensing time of the RS-SA.

3.4c Simulation Results

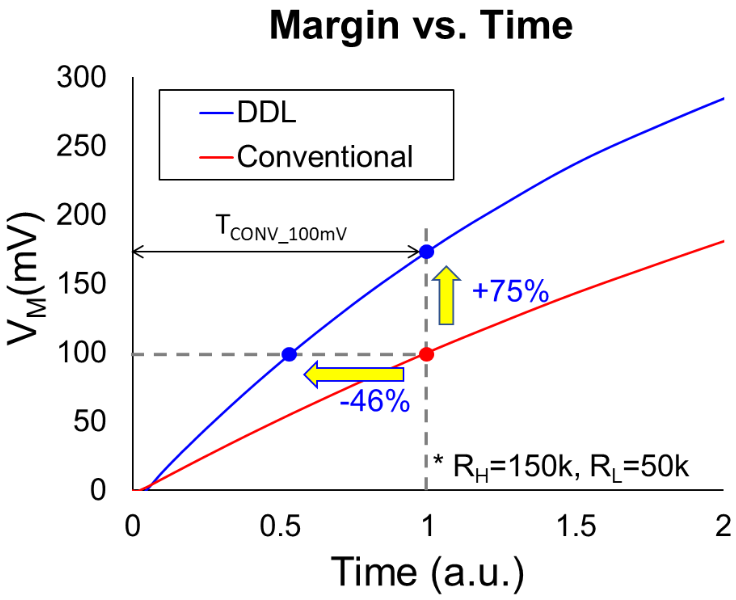


Fig.3.4.4. Read margin as a function of BL development time. For the same 100mV margin target, the DDL scheme can reduce sensing time by 46% or increase margin by 75%.

To verify the performance of the DDL scheme, we again compose the critical path of a 1T1R memory array with layout-extracted parasitics in 28nm. We adopt the sense amplifier of **Fig.3.4.2d** and set V_{READ} to 1V.

Fig.3.4.4 presents the sensing margin waveform of the DDL scheme and the conventional sensing scheme, using a device with R_L=50kΩ and a resistance ratio of 3 [85]. The sensing margin for successful sensing is set to $2V_{M\_CONV} = 100mV$ for the conventional read operation. The time necessary for V_{BL} to develop such a margin is t_{CONV_100mV}. With the DDL scheme, a 75% larger sensing margin (175mV) is achieved at t_{CONV}. For the same target margin ($2V_{M\_DDL} = 100mV$), the DDL scheme reduces the BL development time by 47%.

Fig.3.4.5(a) shows the margin improvement of the DDL scheme for various memory technologies, each with different low/high resistance values (R_L, R_H) and resistance ratios (R-ratio=R_H/R_L). MTJs typically have an R-Ratio between 2-3 and R_L in the range of a few kΩ for STT-MRAMs and 50~100 kΩ for MeRAMs. ReRAM and PCRAM usually have R_L resistance between 10 kΩ~100 kΩ and R-Ratios between 10~100. To compare sensing time and margins on the same scale, we rescale the BL loading of each R_L value so that the time constants are

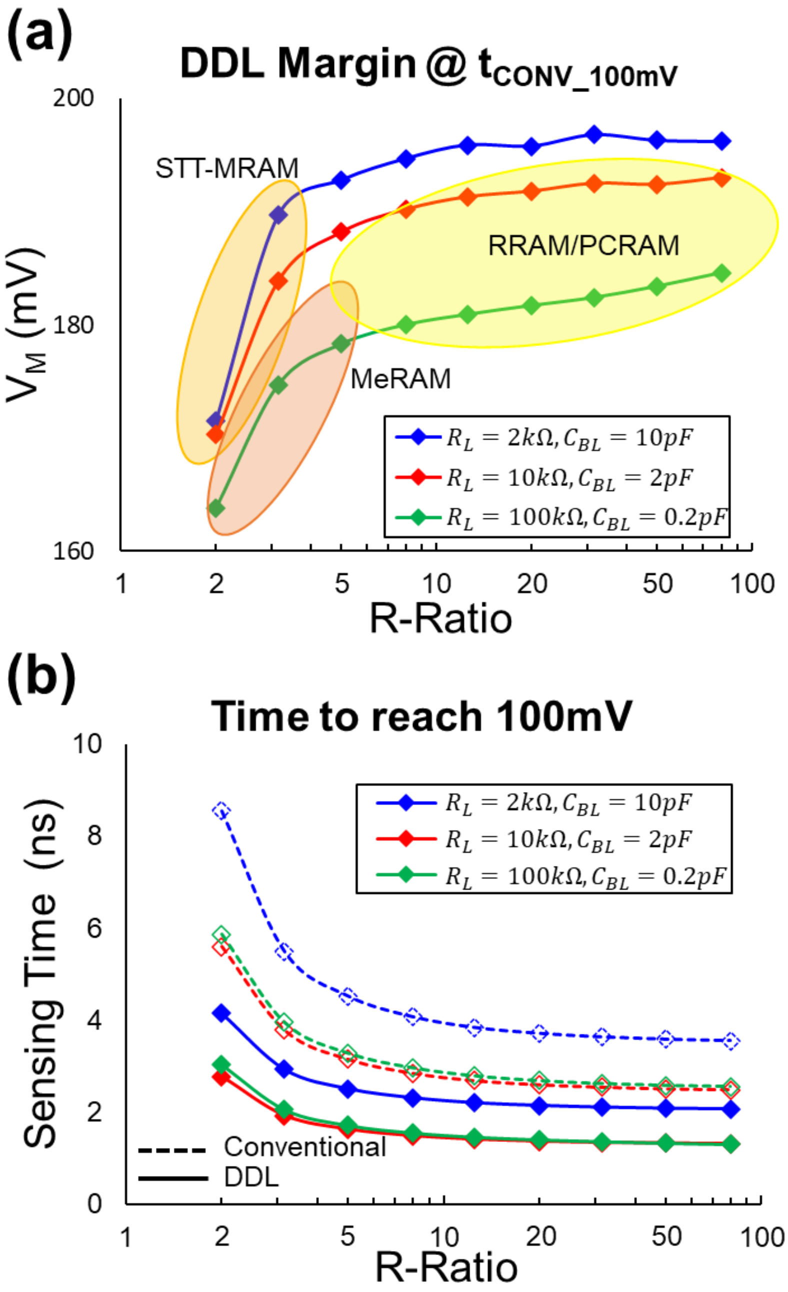


Fig.3.4.5. (a) Read margin and (b) sensing time as a function of R_L and R-Ratio. BL capacitances C_{BL} are scaled so that the time constant for each resistance is the same. The margin improvement increases and approaches 2x for higher R-Ratios. On the other hand, the sensing time improvement is independent of resistance value or R-Ratio and shows a ~50% improvement.

identical. The DDL scheme achieves an average margin improvement of 69%, 89%, and 91% for R-ratios of 2, 10, and 100, respectively. For different $R_L$of 2k, 10k, and 100k, an average margin improvement of 92%, 88%, and 78% can be observed. The margin improvement of the

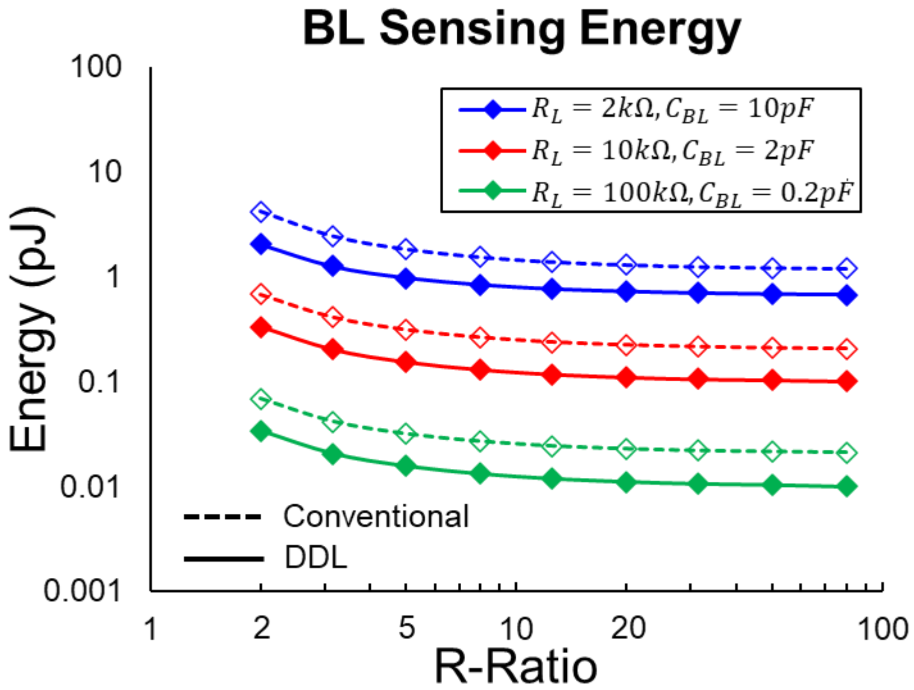


Fig.3.4.6. Energy required for successful sensing of the DDL and conventional scheme. The DDL scheme reduces read energy by ~50% across all R_L and R-Ratios

DDL scheme approaches 2x with a higher R-ratio, in which the exponential decay in Eq.3.4.2 and equation Eq.3.4.5 can be approximated by a linear function.

Fig.3.4.5(b) presents the sensing time (i.e., the time to reach $V_M = 100mV$) under different memory technologies. The sensing time shows an improvement with a higher R-ratio but saturates above R-ratio=10 as the R_L and R_H current difference stays relatively constant. Compared to **Fig.3.4.5(a),** where the margin improvement depends on the resistance and R-ratio, the DDL achieves a technology-agnostic 50% reduction in sensing time as predicted by our theoretical analysis in section 3.4a.

Fig.3.4.6 presents the sensing energy of the DDL scheme compared to the conventional scheme. The sensing energy is extracted as the total energy consumed from VDD which includes dissipation from the read discharge path, the pre-charge/pre-discharge circuitry, and the amplifiers. A slightly below 50% reduction in energy is observed across all cases, in line with the theoretical results predicted by Eq.3.4.6.

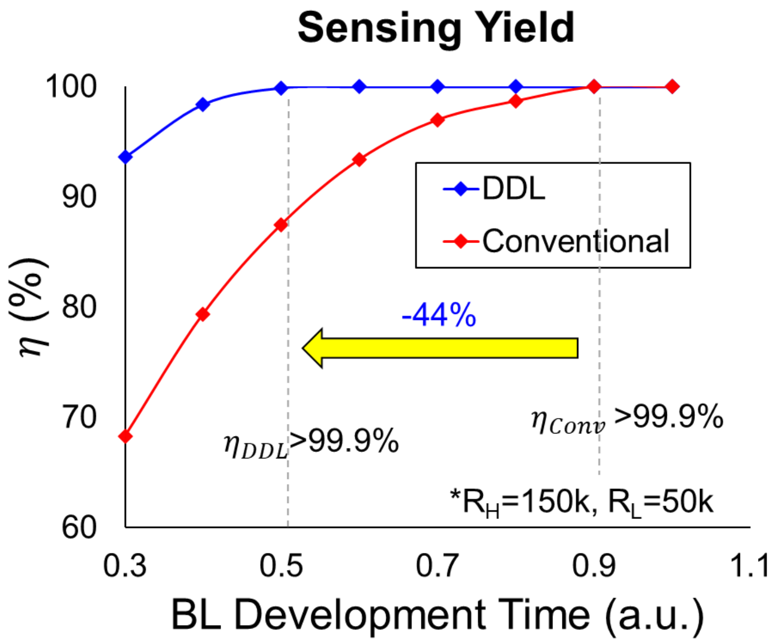


Fig.3.4.7. Sensing yield η as a function of read time for the conventional and DDL schemes. For the same 3-sigma yield requirement (η >99.9%), the DDL scheme can reduce the sensing time by

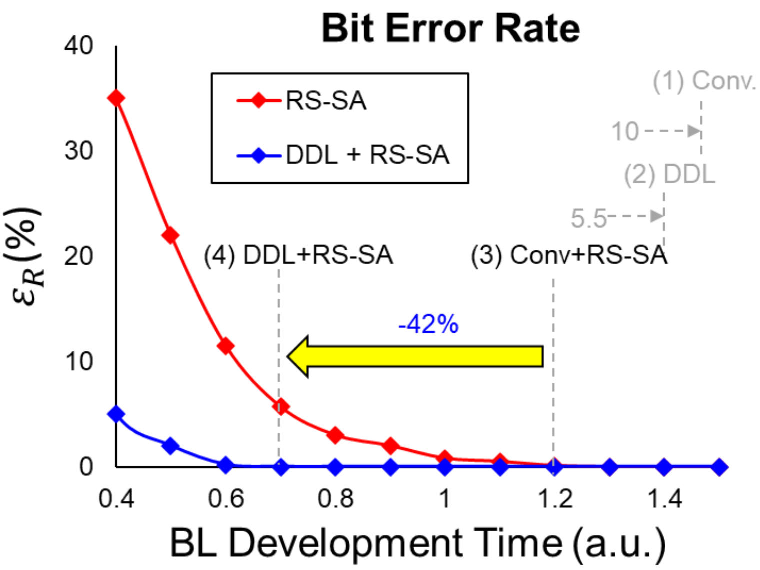


Fig.3.4.8. Read error rate ε_R as a function of sensing time for (1) the conventional scheme, (2) the DDL scheme, (3) conventional sensing with RS-SA, and (4) DDL sensing with RS-SA. The RS-SA improves sensing time by 88%. With DDL, the sensing time is further reduced by an additional 42%.

To translate the DDL performance to reliability and yield η under variation, we conducted Monte-Carlo analysis with local mismatch models. **Fig.3.4.7** presents the read bit error rate ($\varepsilon_R = 1 - \eta$) as a function of BL developing time. We observe that as the sensing time increases, the yield improves for both sensing methods. For a target ε_R of <0.1% or η>99.9%, the conventional sensing method requires a sensing time of 0.9 time units, while DDL can achieve the same yield in 0.5units; representing a 44% reduction.

Finally, as mentioned earlier, the DDL scheme benefits from new, advanced sense amplifier designs. Here, we show the performance of the application of DDL to the RS-SA described in Sec.3.4b. **Fig.3.4.8** displays the ε_R of the following configurations: (1) conventional scheme with the conventional sense amplifier, (2) DDL scheme with the default DDL sense amplifier, (3) conventional scheme with the RS-SA, and (4) DDL scheme with the RS-SA. Comparing (1) and (3), the RS-SA shows a great reduction in sensing time by 88%, which matches the results presented in [190]. Comparing (3) and (4), we observe that the DDL scheme can further reduce the RS-SA sensing time by an additional 42%.

CHAPTER 4

SPINTRONIC SIGNAL PROCESSOR

4.1 Motivation

Over the last 50 years, transistor technology has been the driving force behind an exponential increase in computation power. However, despite its success, transistors face several fundamental limits in computing efficiency:

(1) Standby Energy: Transistors exhibit leakage that consumes power even when it’s not switching. This issue has become increasingly severe in advanced technology nodes as device dimensions scale [191].

(2) Structural Bottleneck: In transistor-based systems, computing and storage are physically separated, which leads to a data transmission bottleneck between the two. This bottleneck, known as the Von-Neumann bottleneck, limits the speed and throughput of the system even as devices scale. While emerging, integrated nonvolatile memories provide a platform to develop non-Von Neumann architectures [34], [105], [167], [192], computing and storage still do not occur in the same device.

(3) Single operation mechanism: Transistors operate on the single mechanism of electron gating. There is only a single function per transistor and there is no correlated functionality among devices. As a result, complex functions are often built upon hierarchies of non-optimized structures.

On the other hand, magnetic devices have no standby energy, and inherently couple storage within their operations. There exists an abundance of spin phenomena for manipulating magnetic states, and devices have been developed with high speed (GHz) and low energy (fJ) while preserving practically unlimited endurance (>10^{15}) [82], [193], [194]. Therefore, there is strong motivation to explore spintronics mechanisms and structures that can fundamentally outperform transistor-based computing. However, today's research focuses primarily on spintronics' application for storage, and few are able to exploit its computing potential as they often fail to outperform the fast scaling CMOS technology.

In this thesis, a spintronic computing device is developed for signal processing that provides orders of magnitude improvement in area, energy, and throughput over state-of-the-art CMOS. Specifically, we realize a convolution engine, a core operation of all major forms of signal processing, including Fourier transforms, filtering, visual and audio processing, machine learning, and many more. However, this operation is extremely costly to compose in CMOS. Two approaches are proposed to realize the spintronic convolution engine, each by coupling a number of spin mechanisms in the device: (a) the Hall effect in combination with domain motion, which has an easy-to-fabricate structure; and (b) magnetic tunneling junctions (MTJs) with domain motion, which provides greater programming-time reconfigurability. All three mechanisms already have strong theoretical foundations and industrial-level fabrication techniques, allowing them to be easily adapted for large-scale production.

In addition to its superior performance, the memory-processing architecture also (1) achieves zero standby power, (2) removes the need to store signal streams, and (3) omits the signal transmission between computing and storage components. Therefore, these devices not only provide high-performance accelerators but also enable efficient signal processing to be adopted in edge and Internet-Of-Things (IOT) devices where large amounts of signal processing were previously too costly to use. In the next section, a review on the background of convolution and the spin mechanisms utilized is presented first.

4.2 Background

4.2a Convolution Operation

A convolution is a mathematical operation on two functions $f(t), g(t)$ to produce a third function $(f * g)(t)$ that resembles the cross-correlation of the two. It is computed by (1) reversing one function: $g(\tau) \rightarrow g(-\tau)$; (2) shifting it: $g(-\tau) \rightarrow g(t-\tau)$; then (3) integrating the product of the shifted, reversed function $g(t-\tau)$ with the other function $f(\tau)$, represented across all shift values, i.e.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau \qquad \ldots Eq. 4.2.1$$

Or in discrete terms,

$$(f * g)[n] = \sum_{-\infty}^{\infty} f[m] g[n-m] \qquad \ldots Eq. 4.2.2$$

Convolutions are the core of many forms of signal processing. In signal filtering, the application of a filter involves the convolution of a kernel with the target signal (such as audio, images, radar, communication, etc). Convolutions are also the main computation for transforms between different domains, such as the Fourier transform and Z-transform, which convolves complex coefficients of phase multiples with a time-domain signal. In machine learning, convolutional layers apply convolution of weights with the layer's input.

4.2b Charge-based Convolution Devices

The convolution operation involves three components: an element-wise multiplication, a summation, and a signal shift. Today, this is implemented digitally; but the prior state-of-the-art hardware was built via charge-coupled devices (CCDs) [195], [196]. This convolution hardware, shown in **Fig.4.2.1**, composes a series of CCD devices with split electrodes. Signals are represented by the analog charge stored in capacitors. The signals are scaled by dividing the charge between the split electrodes; then summed via capacitive coupling to a shared bus. At the end of each cycle, the charge is shifted to the next stage.

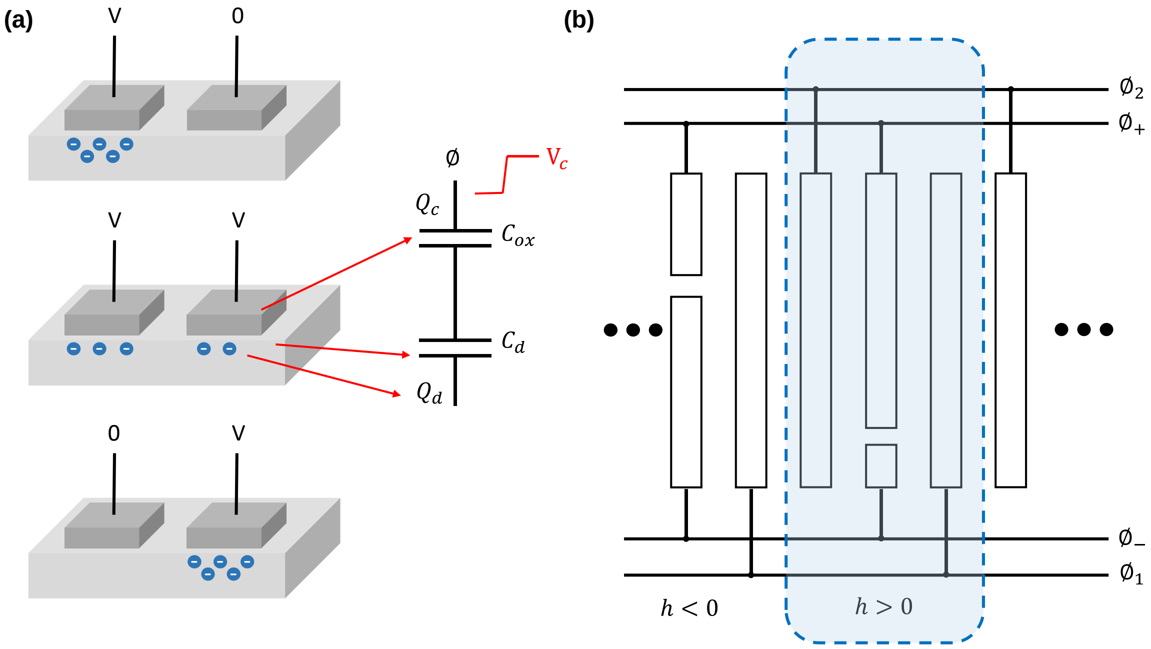


Fig.4.2.1 CCD-based convolution hardware. (a) structure and operation of a CCD. A CCD is composed of depletion region capacitors with electrodes controlling the potential underneath. By consecutively pulsing the electrodes, charge can be moved between the capacitors. Electrically, this can be modelled as the oxide capacitor C_{ox} in series with the depletion capacitor C_d, and a charge Q_d stored in C_d. (b) The CCD-based convolution engine composes a series of three-phase CCDs. Each stage has two isolation electrodes driven by $\emptyset_1$ and $\emptyset_2$, and split electrodes driven by $\emptyset_+$ and $\emptyset_-$. The difference in the size of the two electrodes determines the scaling factor h as described in Eq. 4.2.5. The stored charge Q_d of each stage is divided by the split electrodes and coupled to $\emptyset_+$ and $\emptyset_-$ for summation.

The derivation of the device operation follows. **Fig.4.2.1a** shows the structure of a CCD, which contains an array of linked depletion-region capacitors with electrodes that control the potential of the region underneath. When no voltage is applied to the electrodes, the charge remains in place. On the other hand, by pulsing each electrode consecutively, the charge can be shifted between the capacitors according to the phase of the electrodes. The CCD is electrically modeled by an oxide capacitor C_{ox} in series with the depletion capacitance C_d, and a charge Q_d stored in C_d. When a voltage step of V_C is placed upon the electrode (via a clock line ∅), the charge flowing to the electrode Q_C is the sum of that necessary to provide the voltage step and that to balance Q_d [196]:

$$Q_C = V_C \frac{C_{ox} C_d}{C_{ox} + C_d} + Q_d \frac{C_{ox}}{C_{ox} + C_d} \qquad \dots Eq. 4.2.3$$

In high resistivity substrate materials, $C_{ox} \gg C_d$, and Eq.4.2.3 can be approximated as

$$Q_C = V_C C_d + Q_d \qquad \dots Eq. 4.2.4$$

Equation 4.2.4 provides a mechanism for readout: at the rising edge of the clock line $\emptyset$, the charge flowing on the clock lines is proportional to that stored in the depletion capacitors.

The convolution device (**Fig.4.2.1b**) composes "stages" of three-phase CCDs, each with two electrodes for isolation and one split electrode for readout. The isolation electrodes are driven by clock lines $\emptyset_1$ and $\emptyset_2$, and the split electrode by $\emptyset_+$ and $\emptyset_-$. The split electrode divides the electrode into two sub-electrodes with an area of $\frac{1}{2}(1+h)$ and $\frac{1}{2}(1-h)$, respectively; where h is the scaling factor or multiplication coefficient. During readout, Q_C is divided between the sub-electrodes proportional to their area. The charge difference between $\emptyset_+$ and $\emptyset_-$ is, therefore[196]:

$$Q_{C+} - Q_{C-} = \frac{1}{2}Q_C(1+h) - \frac{1}{2}Q_C(1-h) = (V_C C_d + Q_d)h \quad \dots Eq. 4.2.5$$

The first term $V_C C_d h$ is an input-independent signal that can be removed via circuit techniques. The second term $Q_d h$ computes a scaled multiplication of the and the CCD charge Q_d, and provides the multiplication component of the convolution.

The readout charge of all CCD devices connected to the same clock line will be summed via KCL, e.g. the total charge on the clock line is equal to the sum of the coupled charge of each connected CCD. The total differential charge in each cycle is thus:

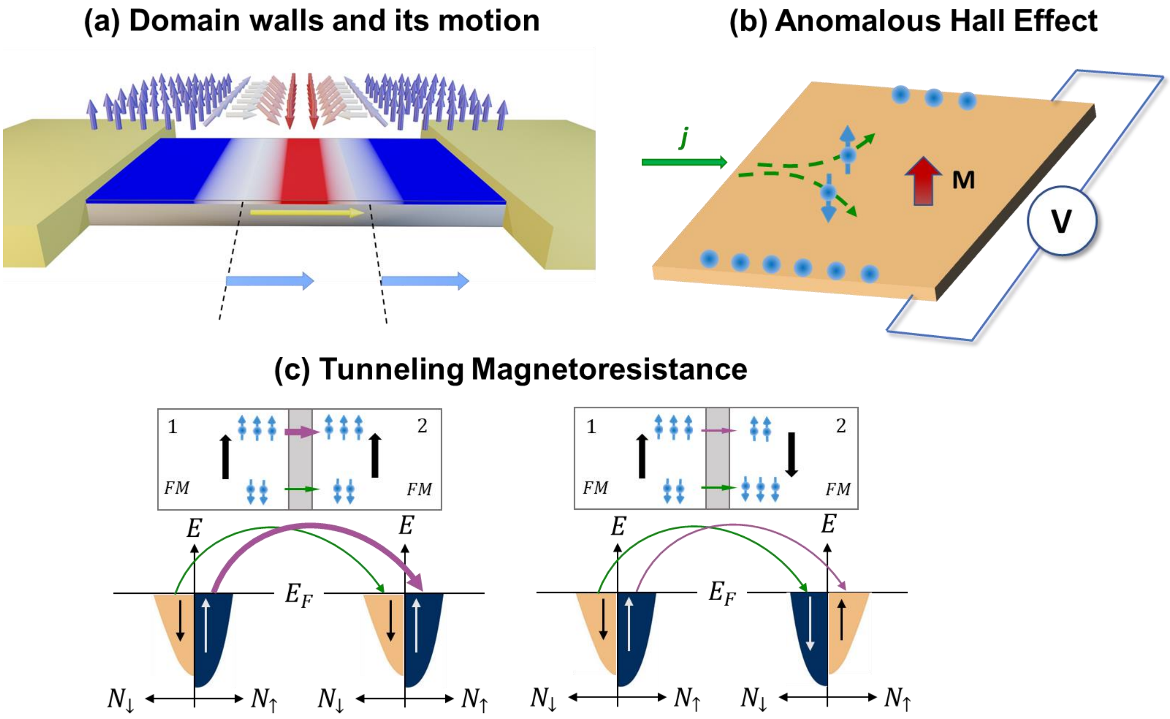
$$Q_{C+,total} - Q_{C,total-} = \sum_{n=1,.,k} (V_C C_d + Q_d[m])h[m] \quad \dots Eq. 4.2.6$$

Where m represents the CCD stage number and k is the total number of CCD stages. At the end of each cycle, the charge in each stage is shifted to the next, resulting in the charge equation taking the form of a convolution:

$$(Q_{C+,total} - Q_{C,total-})[n] = \sum_{k} (V_C C_d + Q_d[m-n])h[m] \quad \dots Eq. 4.2.7$$

While charge-based convolution hardware outperformed other technologies in the 1980s, they eventually fell out of favor as they face the following limitations: (1) scalability limitations as the amount of stored charge Q_d decreases with device scaling; (2) maximum computing stage limitations due to incomplete charge transfer and leakage; and (3) speed limitations due to the capacitance-based computing mechanism.

4.2c Spintronic Mechanisms for Computing


(a) Domain walls and its motion

(b) Anomalous Hall Effect

(c) Tunneling Magnetoresistance

Fig.4.2.2 Spin-based mechanisms used to build the spintronics convolution engine. (a) Magnetic domains (DMs) are regions where magnetic moments are aligned. Domain walls (DWs) are formed between adjacent domains with different magnetization orientation. DW motion can be induced via a spin-transfer from a current through the DW. (b) The Hall effect, where a lateral electric field forms in a current-carry conductor placed in a magnetic field proportional to its strength. (c) The tunneling magnetoresistance effect: in a tunneling junction, carriers can tunnel through the barrier with a probability depending on the magnetic orientation of the two layers.

Our magnetic signal processing engine overcomes the limitations of charge-based convolutions by controlling spin mechanisms to implement multiplication and shifting. In this section, we briefly introduce the spin mechanisms that we utilize.

Magnetic Domains and their motion

In a magnetic material, a magnetic domain (DM) (**Fig.4.2.2a**) is a region where the magnetization of the atoms is stable and aligned towards the same direction. This phenomenon was first formulated by Pierre-Ernest Weiss in 1906, who theorized that adjacent atoms experience a strong effective field from their neighbors in a magnetic material, causing their alignment.

Adjacent domains have different, often opposite, magnetic orientations. In the transition region between the two domains, the magnetic moments rotate smoothly between the two orientations. This forms a spiral spin texture called a domain wall (DW). Consequently, DWs can be created by forming oppositely magnetized domains via switching mechanisms such as spin-orbit-torque (SOT) or magnetic field [197]–[200].

When sufficient current flows through a domain wall, spin transfer from electrons to the DW can displace it [201], [202]. The amount of displacement depends on the total amount of spins transferred, and therefore proportional to the current density and the duration of the pulse. In a magnetic strip where there is only a single current path, all DWs in the strip will be displaced by the same amount, leading to a shifting of the entire magnetic pattern on the strip.

DWs and DMs are the core characteristics to hard drives and racetrack memory [62], [203]. They are responsible for storing data in the orientation of domains and shifts domains to address individual bits. The former shifts domains mechanically via a rotating disk, while the latter shifts domains electrically via current-induced DW motion.

Anomalous Hall effect

The Hall effect (**Fig.4.2.3b)** refers to the phenomena in which a lateral electric field is created across a current-carrying conductor placed in a perpendicular applied magnetic field. First discovered by Edwin Hall in 1878 [204], this phenomenon comes from the balance of two forces acting on the moving charged particles: the Lorentz force and the Coulomb force. In a material with electrons as the dominant charge carrier; the Lorentz force is described by

$$F_{Lorentz} = qv \times B \quad \ldots Eq. 4.2.8$$

where q is the unit electron charge, v is the speed of the electron, and B is the magnetic field. Under the influence of this force, charge carriers are pushed towards the edges of the material. Without a discharge path, these carriers accumulate and create an electric field that exerts a Coulomb force on subsequent charge:

$$F_{Coulomb} = qE \quad \ldots Eq. 4.2.9$$

, where E is the generated electric field. The carriers will continue to accumulate until the two forces cancel out in the steady state:

$$F_{total} = q(E + v \times B) = 0 \quad \ldots Eq. 4.2.10$$

In a device where charge flows in the $x -$direction with speed v_x and the magnetic field is perpendicular B_z, the lateral electric field E_y can be obtained as

$$E_{Hall} = E_y = v_x B_z \quad \ldots Eq. 4.2.11$$

The Hall effect plays an important role in material characterization and is frequently used to measure carrier concentration in conductors and solid-state electronics. In 1881 [205], Hall further reported that this effect is ten times greater in a magnetized ferromagnetic iron than in non-magnetic conductors. Known as the Anomalous Hall Effect (AHE), this phenomenon depends on the magnetization M of the material and is also used to study the magnetic properties of ferromagnetic materials.

Tunneling Magnetoresistance effect

A magnetic tunnel junction (MTJ) composes two ferromagnets separated by an insulating barrier. When the insulating layer is thin (in the single/sub-nanometer range), electrons can tunnel between the ferromagnet layers via the quantum tunneling effect (**Fig.4.2.3c)**. First observed by Michel Jullière in 1975 [43], the tunneling probability is dependent of the relative orientation of the two ferromagnets. When the magnetic orientation of the two ferromagnetic layers are parallel to each other (P state), the tunneling probability is high, and when the magnetic orientations are anti-parallel (AP state), the tunneling probability is low. The difference in tunneling probability gives rise to a change in the electrical resistance of the MTJ. One way to describe this effect is via the tunneling magnetoresistance ratio (TMR), defined as:

$$TMR = \frac{R_{ap} - R_p}{R_p} \quad \ldots Eq. 4.2.12$$

Where R_{ap} is the MTJ resistance in the anti-parallel state and R_p is that of the parallel state. The ability to convert magnetic orientations to an electrically accessible resistance makes TMR a useful readout mechanism for magnetic sensors and magnetic random access memory (MRAM).

4.3 Anomalous Hall -Domain Wall Motion(AHE-DWM) Convolution Engine

4.3a Device Structure

In this section, we describe the device structure and operation of the proposed spintronic convolution engine. We first show how multiplication is achieved through the combination of DMs and the Hall effect and magnetic domains, followed by the summation circuitry and DW motion to complete the convolution.

Multiplication Unit

The multiplication unit of the processor (**Fig.4.3.1a**) has the structure of a modified Hall bar device. The body of the device is a domain-hosting magnetic strip of width W_S and length L_S. Two gold electrodes separated laterally by D_E are placed on top of the strip. Each electrode has a width of W_E and length of L_E.

During computation, a domain is shifted within the electrode region. The domain has a length of L_D pointing in the $+z$ direction. As a result, a portion of the region within the electrode (L_D) has magnetization pointing in the $+z$ direction, and the remainder of the

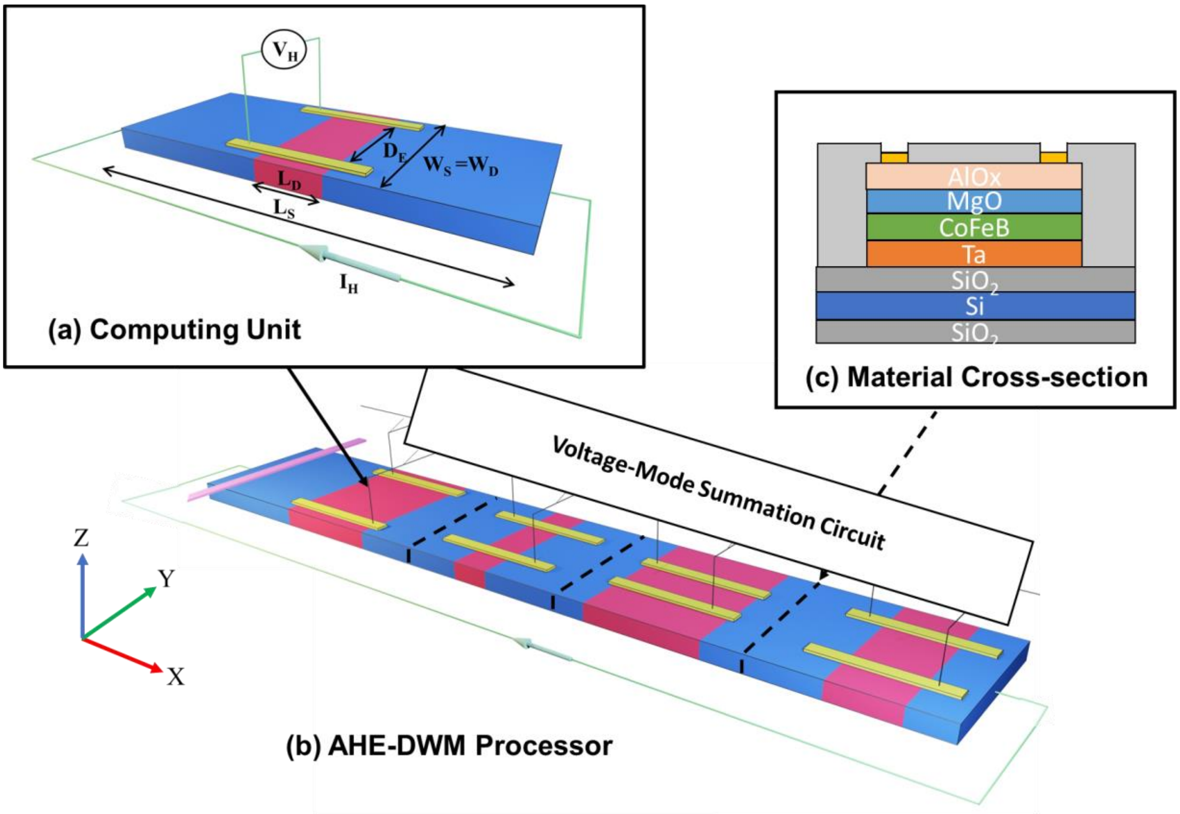


Fig.4.3.1 AHE-DWM based convolution engine. (a) the computing unit, with the structure of a domain-hosting magnetic strip and electrodes placed on top. The computing unit outputs the Hall voltage as a product of the domain length L_D and the distance between the hall electrodes D_E. (b) the convolution engine composes a series of multiplication computing units that shift the domain signals between each computing unit. Multiplication results are summed using low power voltage summation circuitry such as switch capacitors. (c) material cross-section of the convolution engine.

electrode $(L_E - L_D)$ pointing in the $-z$ direction. To the first order, the net perpendicular magnetization in the electrode region $M_{E,z}$ is the sum of the $+z$ and $-z$ pointing components:

$$M_{E,z} = \; L_D M_s - (L_E - L_D) M_s \qquad \dots Eq. 4.3.1$$

Where M_s is the magnetic strip's saturation magnetization. Combining Eq.4.3.1 and Eq.4.2.11, the AHE electric field of the electrode region can be represented as

$$E_{E,y} = C I_x M_{E,z} \qquad \dots Eq. 4.3.2$$

where the current through of the strip I_x is proportional to qv_H and C is a constant related to the AHE coefficients as well as material and device constants. The Hall voltage is the Hall electric field multiplied by the distance between the Hall electrodes:

$$V_{E,y} = D_E E_{E,y} \qquad \dots Eq. 4.3.3$$

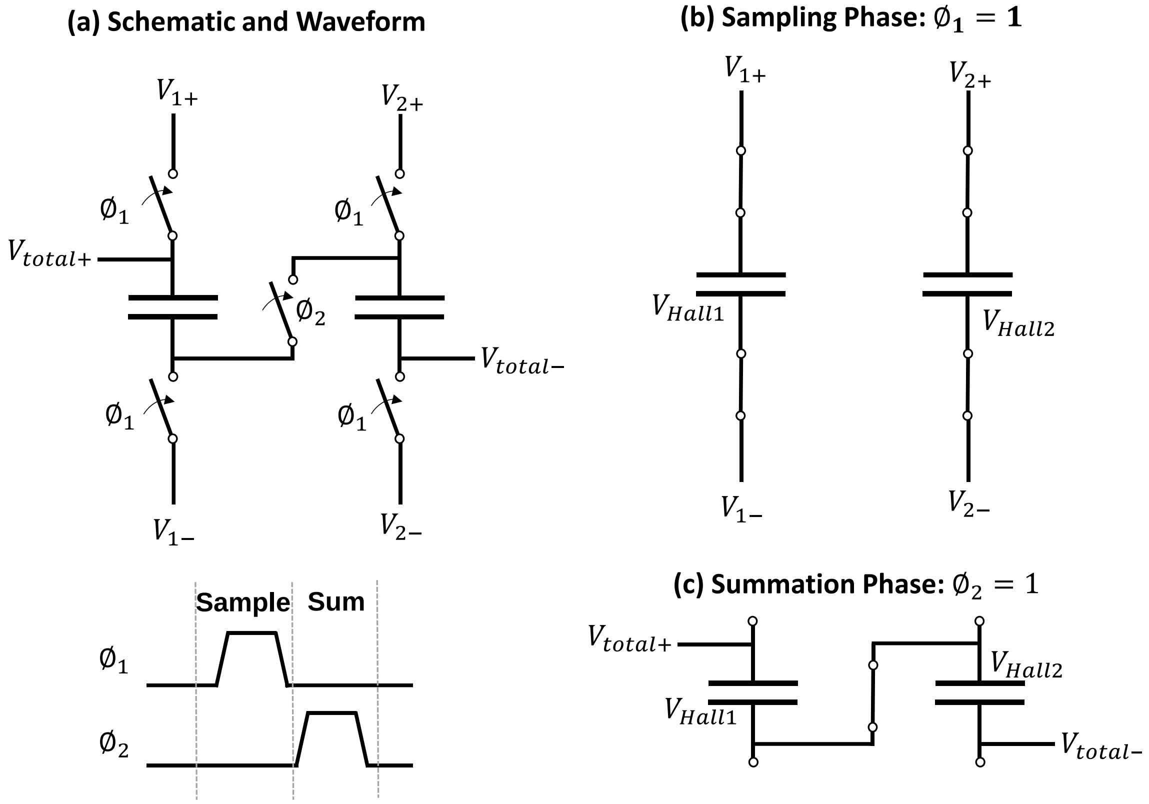


Fig.4.3.2 Example of a switch-capacitor circuit used to sum voltages. (a) schematic and operation waveform. The circuit operations in two phases. (b) In the sampling phase, the capacitors are connected to the Hall electrodes, and the Hall voltages V_{Hall} of each multiplication unit is sampled on the capacitors. (c) In the summation phase, the capacitors are connected in series to sum V_{Hall}.

Combining equations 4.3.1 to 4.3.3 and separating the input-independent components, we arrive at the following representation for the Hall voltage:

$$V_{E,y} = C_1 + C_2 J_x L_D D_E \ldots Eq. 4.3.4$$

Where C_1, C_2 composes of design and material constants. $Eq.$ 4.3.4 provides the desired multiplication function: under a given read current, the AHE voltage is the linear product of the domain length l_D and the distance between the electrodes d_E.

Convolution Engine

The convolution engine (**Fig.4.3.1b)** composes a series of multiplication units, a summation circuit, and a domain write circuit. During each computing cycle, the AHE voltages of each multiplication unit are summed using a low-power analog summation circuitry. An example of the circuitry is the switch-capacitor circuit shown in **Fig.4.3.2**. It operates in two phases: In the first phase, the two terminals of each sampling capacitor are connected to the Hall electrodes to sample the Hall voltages. In the second phase, the capacitors are switched to connect in series to sum the sampled voltages. The output of the summation circuitry in each cycle is:

$$V_{E,total} = NC_1 + C_2 \sum_{n=0}^{N} L_D[n]D_E[n] \qquad \ldots Eq. 4.3.5$$

where n is the stage of the multiplication unit and N is the total number of multiplication units. Inverting the input signal sequence $L_D[n]$ to $L_D[-n]$ and shifting domains to the adjacent multiplication unit at the end of each cycle, the output voltage as a function of the cycle number m takes the exact form of a convolution:

$$V_{E,total}[m] = NC_1 + C_2 \sum_{n=0}^{N} L_D[m-n]D_E[n] \quad \ldots Eq. 4.3.6$$

The write circuitry is used to program inputs to the strip. In our design, it composes a field-generating wire, which switches the polarity of the domain beneath it. The field is controlled by a pulsed current that determines the length of the input domain.

4.3b Performance Analysis

In this section, we theoretically derive the performance of the proposed spintronic convolution engine. We show material limits and estimate the device area, throughput, and energy on a $28nm$ process; then compare its performance with a CMOS implementation at the same technology node.

Device dimensions

The length of the multiplication unit's electrodes has to cover the domain signal L_D, which is bound by the minimum size of a magnetic domain. This size is limited by the free energy to nucleate a domain, which can be written as the domain wall energy under the influence of the demagnetization field and the external field:

$$E = 2W_D\sigma t - u_0 {M_s}^2 W_D L_D t - 2u_0 H_C M_s W_D L_D t \quad \dots Eq. 4.3.6$$

Where W_D and L_D are the domain width and length, t is the film thickness, σ is the domain wall energy (J/m^2), and H_C is the coercivity (A/m) of the film. The condition for a domain to be stable is that a change in the domain state would result in higher energy. In other words, the partial derivative of the energy with respect to the domain width $\frac{\partial E}{\partial W_D}$ should be smaller than or equal to 0. Applying this to Eq.4.3.6, we arrive at the minimum size for a domain to exist:

$$L_D \geq 2\frac{\sigma}{u_0 {M_s}^2 + 2u_0 H_C M_s} \qquad \dots Eq. 4.3.7$$

In addition, for the domain to be stable under thermal fluctuations, its thermal stability should exceed a target thermal stability:

$$\Delta = \frac{E}{K_b T} > \Delta_{target} \qquad \dots Eq. 4.3.8$$

Where K_b is the Boltzmann constant, T is the temperature, and Δ is the thermal stability. The requirement for Δ is 30 for computing and 40 for storage applications. With the material parameters of a Ta/CoFeB/MgO system shown in **Table 4.3.1** [55], [206]–[209], the minimum domain length $L_{D,min}$ is $8.2nm$.

The exact length of a multiplication unit also depends on the resolution of the input signal. When $L_D = L_E$, the magnetization of the region is fully in the $+z$ direction, representing the maximum input value. When $L_D = 0$, the magnetization of the region is fully in the $-z$ direction, representing the minimum input value. To have n-bit input resolution within this range, each input step corresponds to an increase in domain length of $\Delta l = L_E/2^n$. As a result, finding the physical limit to Δl would give the requirement for the device length.

In theory, Δl can be a few atoms in length, while in devices it will depend on (1) the granularity of domain creation, and (2) pinning centers in the magnetic strip. When domains

Table 4.3.1 Device and material parameters of a Ta/CoFeB/MgO system

Symbol	Definition	Value
$\boldsymbol{\sigma}$	**Domain Wall Energy**	$\mathbf{5.3427}\ \ \boldsymbol{mJ/m^2}$
$\boldsymbol{M_s}$	**Saturation Magnetization**	$\mathbf{10^6}\ \ \boldsymbol{A/m}$
$\boldsymbol{H_C}$	**Magnetic Coercivity**	$\mathbf{2.4\times10^4}\ \ \boldsymbol{A/m}$
$\boldsymbol{\delta}$	**DW Speed** **(Min, Max)**	$\mathbf{5\times10^{-4}}\ \ \boldsymbol{m/s}$ $\mathbf{20}\ \ \boldsymbol{m/s}$
$\boldsymbol{J_{Shift}}$	**DW Current Density** **(Min, Max)**	$\mathbf{1.5\times10^8}\ \ \boldsymbol{A/m^2}$ $\mathbf{7.5\times10^{10}}\ \ \boldsymbol{A/m^2}$
$\boldsymbol{J_{Hall}}$	**Hall Current Density**	$\mathbf{1.2\times10^8}\ \ \boldsymbol{A/m^2}$
$\boldsymbol{l_{min}}$	**Minimum Domain Length**	$\mathbf{8.2}\ \ \boldsymbol{nm}$

are created by a current pulse (such as using a current induced field), Δl is the product of the domain speed δ and the minimum pulse width p_{min}:

$$\Delta l = \delta p_{min} \qquad \ldots Eq. 4.3.9$$

In a $28nm$ technology, the minimum stable pulse-width p_{min} is ~10ps, which corresponds to Δl of 0.2nm. Finally, the minimum length is limited by the fabrication feature size. Putting together the above limitations, the dimensions L_E can be obtained as for a binary multiplication unit is $28nm$, while that for an 8-bit unit is $53nm$.

Similarly, the constraint to the minimum width of the multiplication unit W_S also follows the argument of domain stability and input granularity. The former requires $\frac{\partial E}{\partial L_D}$ to be smaller than or equal to 0. On the other hand, the latter depends on the fabrication process of the distance between the electrodes. From the device structure in **Fig.4.3.1**, it is evident that

$$W_S > D_E + 2W_E + 2^n \Delta W \qquad \ldots Eq. 4.3.10$$

where W_S is the width of the strip, W_E is the width of the electrode, D_E is the distance between the electrodes, n is the input resolution, and Δw is the minimum step of the fabrication process. In a 28nm technology, the minimum values for D_E and W_E are both the fabrication feature $F = 28nm$, and the minimum change in distance ΔW is $1nm$. This corresponds to $W_S = 84nm$ for binary and $W_S = 340nm$ for 8-bit multiplication. It should also be noted that in the AHE-

DWM processor, the area of the magnetic strip is not the dominating factor. Rather, the sampling circuitry takes up the majority of area with switches and capacitors.

Operation Frequency

The bottleneck to the convolution engine's frequency is the larger delay of the two operations: (1) sampling and summing the AHE voltages of each multiplication unit, and (2) writing the next domain into the strip and shifting existing domains to the adjacent multiplication unit. Each operation corresponds to a different phase in the clock cycle.

The delay of sampling and summation operation composes (a) the delay of charging the sampling capacitors and (b) the delay of the switch capacitor circuit for summation. The former can be computed via the relationship $t_c = CV/I$, where t_c is the charge time, C is the capacitance, V is the voltage, and I is the charging current. The latter is limited by the intrinsic switching delay of CMOS transistors on the order of a few to tens of ps. With CMOS capacitors of ~1fF, Hall voltages on the order of tens of mV, and charging currents in the uA~mA range, the delay of this phase is in between ten to a few hundred ps.

The delay of the write and shift operation is the time necessary to drive domains from one computing unit to the next. This can be calculated as the length of each unit plus the minimum separation, divided by the speed of the domain wall δ, e.g., $(L_E + F)/\ \delta$. For Ta/CoFeB/MgO materials, this phase is the main limiting factor. With in the dimensions earlier, we can obtain that a binary convolution engine in $28nm$ has a delay of 2.7ns (183MHz) and that for an 8-bit system is 3.9ns (128MHz).

Energy Consumption

During the sampling and summation phase, the energy includes: (1) the read current in the magnetic strip, (2) the charging of the sampling capacitor, and (3) the switching of four switches in the switch-capacitor circuitry. This can be expressed as

$$E_{Read} = I_{Hall}^2 R_s t_1 + C_{sample} V_E^2 + 4 C_{sw} V_{sw}^2 \qquad \ldots Eq.\, 4.3.11$$

Where C_{sample} and C_{sw} are the capacitance of the sampling capacitors and switches, V_E is the output Hall voltage, V_{sw} are the switches operating voltage, I_{Hall} is the current in the magnetic strip, R_s is the resistance of the strip, and t_1 is the period of the sampling and summation phase. Using the dimensions derived in the previous section and the parameters in Table 4.3.1, we can obtain $I_{Hall} = 1uA$ and $R_s = 200\Omega$. With C_{sample} and C_{sw} of 1fF and $V_{sw} = 0.6V$, we can obtain an energy of $2.4 \times 10^{-17}J$ for binary multiplication and $7.9 \times 10^{-17}J$ for 8-bit multiplication.

During the shift and input phase, the energy consumption includes: (1) the current to shift domains to the next computing unit, and (2) the current to write the next domain signal to the beginning of the strip. This can be expressed as:

$$E_{shift} = I_{shift}^2 R_s t_2 + I_{field}^2 R_{field} t_2 / N \quad \dots Eq. 4.3.12$$

where I_{shift} and I_{field} are the current necessary for shifting domains and generating the switching field; R_{field} is the resistance of the field-generating wire, and t_2 is the period of the shift and input phase. Note that the energy for generating the switching field is divided by N, the number of multiplication units, as only one input is being written per cycle for the entire engine. Using the parameters in Table 4.3.1, we can obtain $I_{shift} = 40uA$. With a wire of $R_{field} = 100\Omega$ and $I_{field} = 100uA$, we can obtain the energy of the shift and input phase to

Table 4.3.2 Performance of the spintronic convolution engine (per multiplication unit).

Metric	Binary	8Bit
l_E	28 nm	53 nm
w_s	84 nm	340 nm
r_{strip}	200Ω	73Ω
t_{Sample}	20 ps	20 ps
t_{Shift}	2.7 ns	3.9 ns
E_{Sample}	3.6×10^{-16} J	3.6×10^{-16} J
E_{Shift}	8.0×10^{-16} J	6.7×10^{-15} J
E_{Write}	2.7×10^{-15} J	3.9×10^{-15} J

be $2.7 \times 10^{-15} J$ and $3.9 \times 10^{-15} J$ for binary and 8-bit multiplication, respectively. A summary of the device dimensions, speed, and energy is shown in **Table 4.3.2**.

A comparison of the proposed spintronic convolution engine and the traditional CMOS-based implementation is shown in **Table 4.3.3**. We compare the performance of a 1024-point convolution hardware with binary and 8-bit resolution. For the proposed design, the energy is $1.2pJ$ per 1024-point binary convolution and $7.2pJ$ per 8-bit convolution. A CMOS implementation in the same technology would require $6.8pJ$ and $132pJ$, respectively. The CMOS design would further necessitate storage and transmission of input data, leading to additional and area overheads not accounted for in this comparison. The throughput of the spintronic engine is lower than CMOS designs at binary resolution (183MOp/s vs 238MOp/s) but better at 8-bit resolution (128MOp/s vs 29MOp/s). The spintronic engine area is significantly smaller than the CMOS design, with an improvement around 100x and 1000x for binary and 8-bit convolutions. Note that the area for the spintronic convolution engine is dominated by the sampling circuitry (5T/cell or 200F^2/cell), which accounts for over 90% of the area in a binary convolution engine and over 80% in the 8-bit design. Overall, the spintronic convolution engine presents 3 orders of magnitude improvement in the throughput/area-energy metric on binary convolution and 6 orders for 8-bit convolution.

Table 4.3.3 Performance of an 1024-point convolution of the proposed and traditional approach.

Quantity	**Proposed**		**CMOS**	
	Binary	**8Bit**	**Binary**	**8Bit**
Area (A)	$167\ \mu m^2$	$191\ \mu m^2$	$2.0 \times 10^4 \mu m^2$	$4.5 \times 10^5 \mu m^2$
Throughput (T)	$183\ MOp/s$	$128\ MOp/s$	$238\ MOp/s$	$29\ MOp/s$
Energy (E)	$1.2 \times 10^{-12} J$	$7.2 \times 10^{-12} J$	$6.8 \times 10^{-12} J$	$1.3 \times 10^{-10} J$
FOM = T/EA	1.8×10^{30}	1.9×10^{29}	3.5×10^{27}	9.6×10^{23}

4.3c Experiments

Verification of the physical mechanism

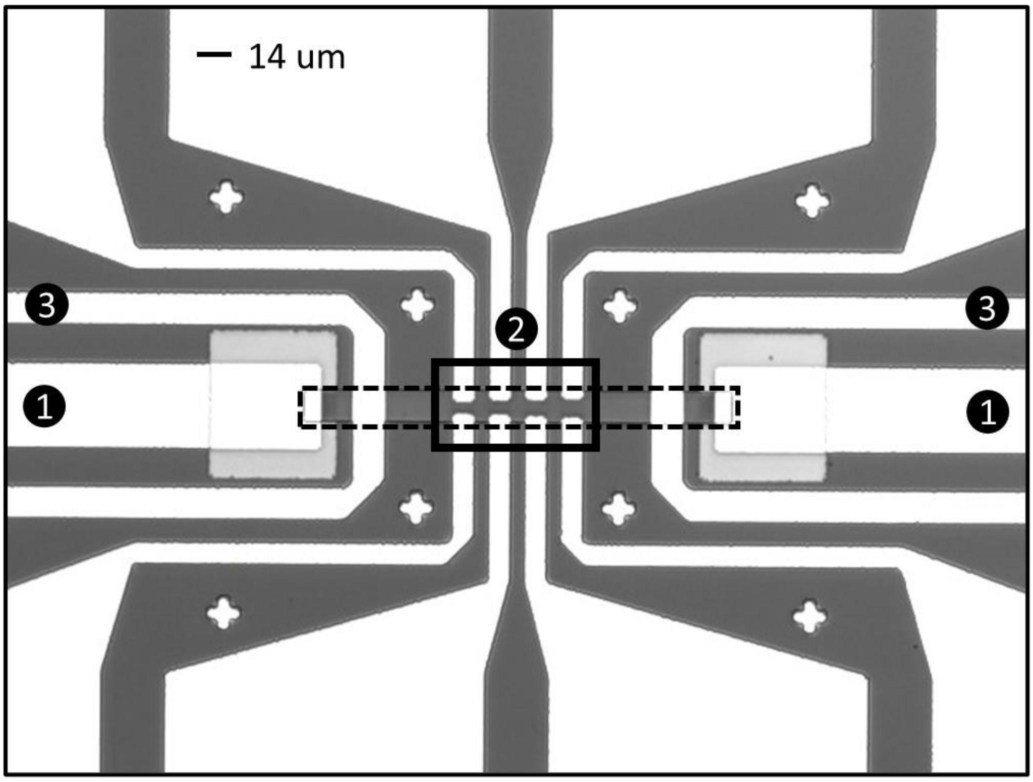


Fig.4.3.3 Experiment setup for extracting the relationship between V_{AHE} and I_x as well as D_E. We fabricate a convolution engine with 4 multiplication units using the Ta/CoFeB/MgO stack. The dashed-line box shows the magnetic strip and the solid-line box shows the convolution engine. (1) represents the current channel where I_x is applied, (2) represents the Hall electrodes/channels where V_{AHE} is measured, and (3) represents the wire for switching magnetic domains. (Figure Credits: Bingqian Dai)

In this section, we describe the experiments to verify the functionality of our proposed convolution device. To verify the multiplication unit, we need to experimentally confirm Eq.3.4.4. This requires obtaining the relationship of the AHE voltage V_{AHE} to the read current I_x (Experiment-1); to the distance between the electrodes D_E (Experiment-2); and to the domain length L_d (Experiment-3). Subsequently, we need to ensure that the output each multiplication unit is not disturbed by the patterns in adjacent units (Experiment-4). In addition, to verify the write and shift operation, we confirm domain creation and domain shifting (Experiment-5).

Experiment-1 and Experiment-2 are conducted via the setup in **Fig.4.3.3**. Using a Ta/CoFeB/MgO magnetic film, we fabricate a number of convolution engines with different D_E. The magnetic strip is labelled by the dashed-line box and the convolution engine by the solid-line box. Components of the convolution engine are numerically labelled: (1) the current channel for the strip where I_x is applied, (2) the Hall channel/electrodes where V_{AHE} are measured, and (3) the wire for generating a local field to switching domains. By sweeping the injected current, we can obtain the relationship between the V_{AHE} and I_x. By comparing measurements with different the distance between the AHE probes, we obtain the relationship between the V_{AHE} and D_E.

The $V_{AHE} - I_x$ relationship is shown in **Fig.4.3.4**, where we observe that V_{AHE} is almost perfectly linear to I_x and the curve crosses the origin. We can thus characterize the $V_{AHE} - I_x$ relationship via R_{AHE}, defined as

$$R_{AHE} = V_{AHE}/I_x \quad \dots Eq. 4.3.13$$

We then characterize the $V_{AHE} - D_E$ relationship by plotting R_{AHE} against D_E in **Fig.4.3.5**. Similarly, the expected linear relationship is observed.

For Experiment-3, Experiment-4, and Experiment 5, we fabricate the magnetic film into a two-channel Hallbar structure as shown in **Fig.4.3.6**. The measured R_{AHE} as a function of the number of shifting pulses is shown in **Fig.4.3.7**, and the domain location as a function of the number of pulses is shown in **Fig.4.3.8**. Before the domain is shifted into the Hall channel (pulse number between 0-5), R_{AHE} is independent of the number of pulses, indicating that the Hall voltage is not affected by the region outside the channel. Afterward, around pulses 5-20, the portion of +z -direction domains within the Hall channel is near linear to the number of pulses. In this region, we al so observe a linear relationship between the R_{AHE} and the number of pulses, confirming the linear $R_{AHE} - M$ relationship necessary for multi-bit multiplication. In pulses 20~30 the domain shifts slower and slightly nonlinear due to pinning centers in the

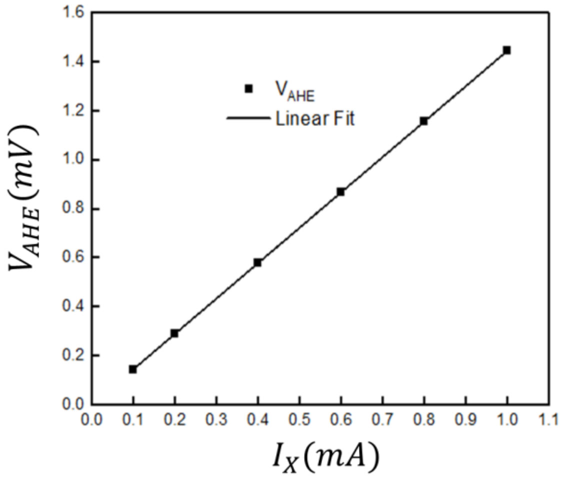


Fig.4.3.4 The $V_{AHE} - I_x$ relationship. Currents between 0.1mA to 1mA are applied in 0.1mA steps. We observe a linear relationship between V_{AHE} and I_x, which can be characterized by R_{AHE}. (Figure Credits: Bingqian Dai)

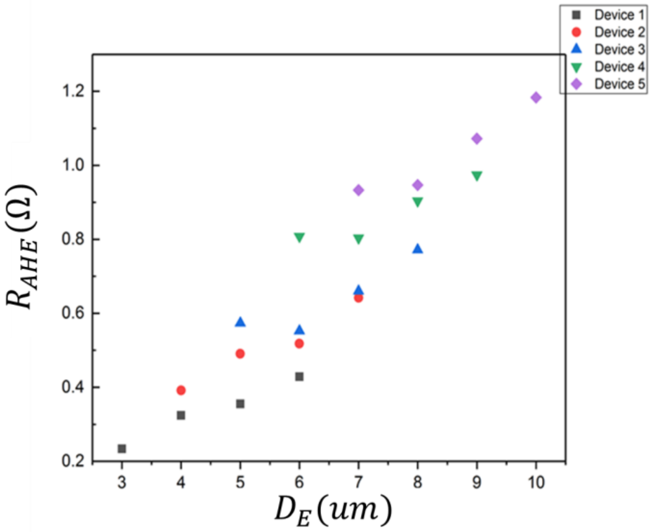


Fig.4.3.5 The $R_{AHE} - D_E$ relationship. The linear relationship confirms the results of Eq.3.4.4 (Credits: Bingqian Dai)

device. Nevertheless, R_{AHE} still shows the desired linear relationship to the total magnetization within the Hall region.

Prototyping

Fig.4.3.9 shows the fabrication process of our magnetic convolution engine: (a-b) On a Silicon substrate, magnetic layers Ta/CoFeB/MgO/Ta are first deposited. (c) Hall electrodes

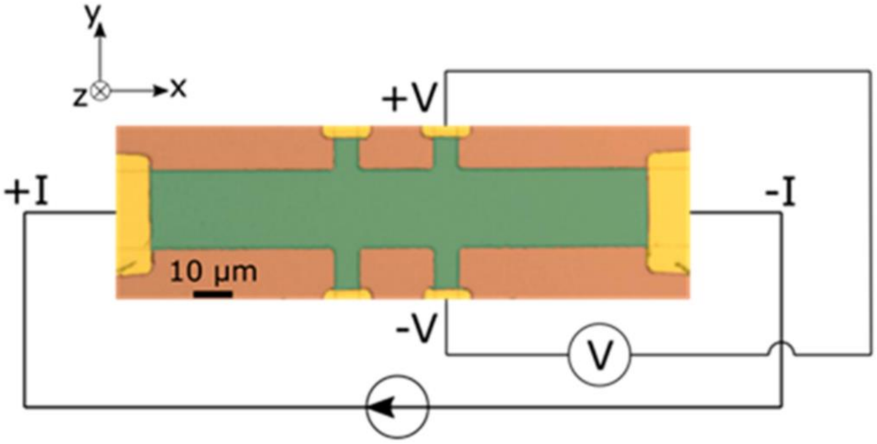


Fig.4.3.6 Experiment setup for the $V_{AHE} - M$ relationship, verification that there is no interference between adjacent devices, and domain shifting. The design consists of a series of two Hallbars, with the AHE voltage measured at the second Hall channel.

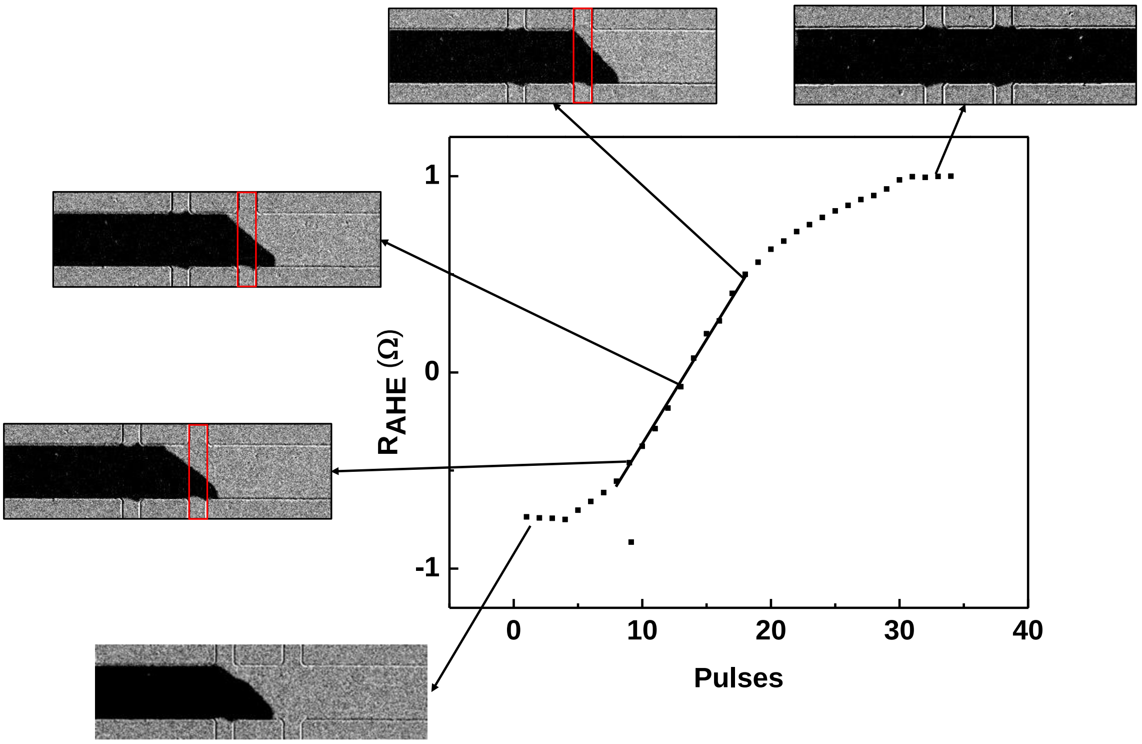


Fig.4.3.7 Measured R_{AHE} as a function of the number of domain-shifting pulses of amplitude 12mA with a pulsewidth of 1ms. Initially, before the domain is shifted into the Hall channel (before pulse 5), the R_{AHE} is constant. Afterwards (pulses 5~20), we observe a linear relationship between R_{AHE} and the number of pulses. At around pulse 20, pinning causes a slowdown in the domain portion, changing the slope between R_{AHE} and pulse number. However, the $R_{AHE} - M$ relationship stays the same.

are then deposited via photolithography to create the photoresist pattern and evaporation to deposit 10nm Au and 5nm Cr. (d-e) The magnetic strip pattern is defined via photolithography

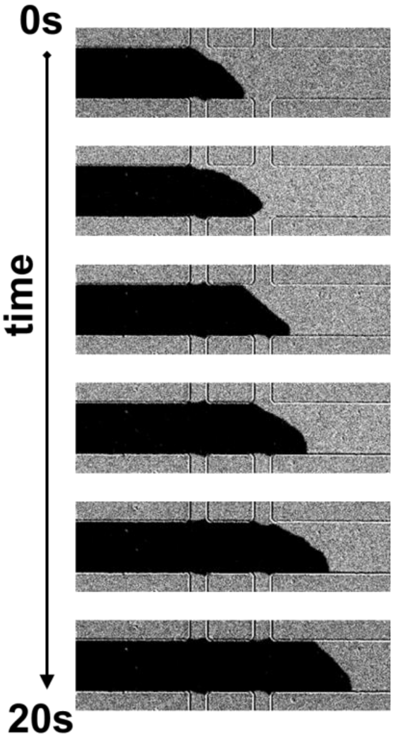
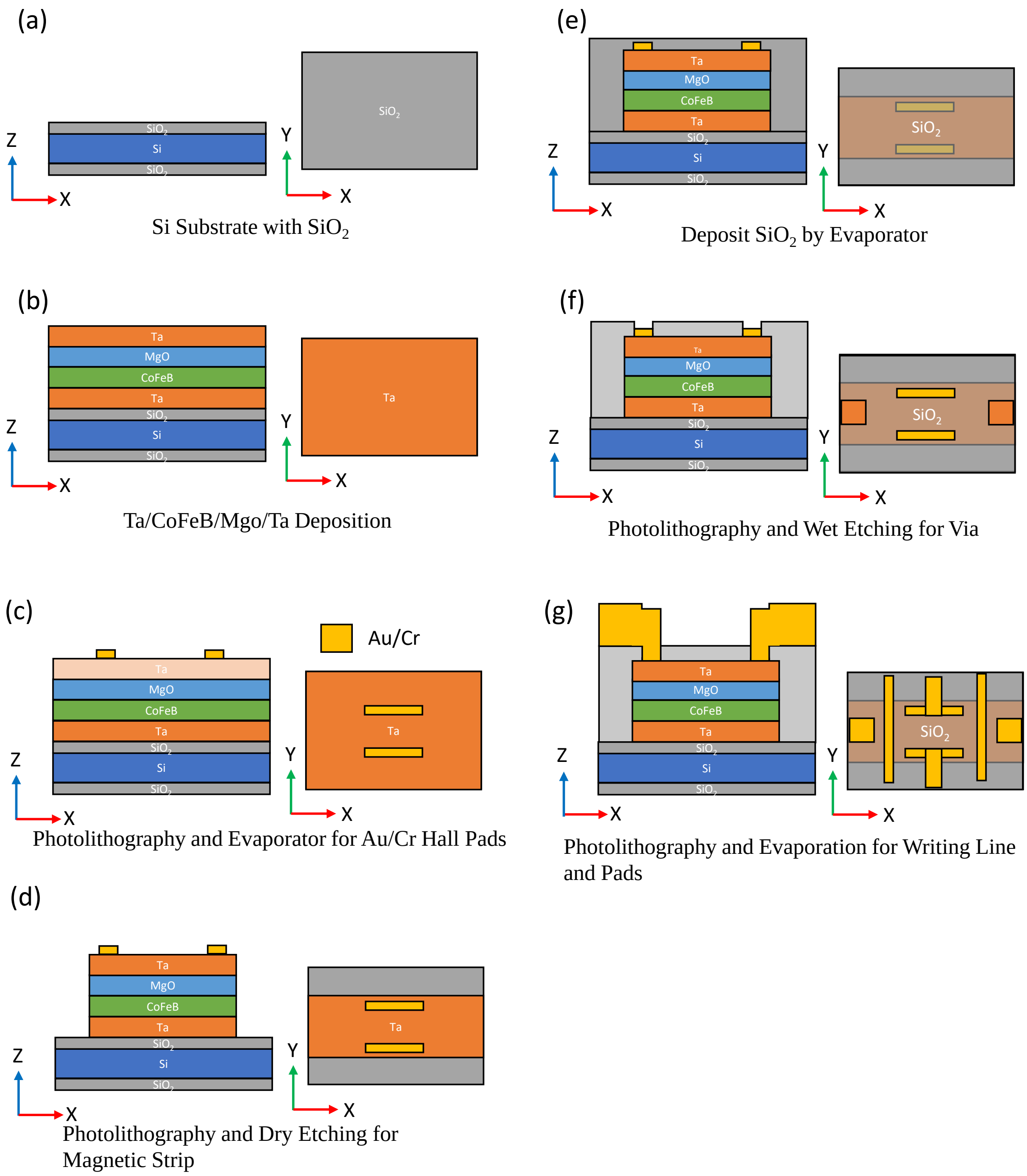


Fig.4.3.8 Measured domain wall motion in the experiment design of **Fig.4.3.6**. Pulses are of 12mA amplitude and 1ms duration and 1s between subsequent pulses.

and dry etching, and insulation (SiO2) is deposited. (f) T he stack is etched until the hall pads are exposed, and (g) photolithography is used to create vias to the hall electrodes and wiring to the external pads. The Completed devices are shown in **Fig.4.3.10**.



Si Substrate with SiO$_2$

Deposit SiO$_2$ by Evaporator

Ta/CoFeB/Mgo/Ta Deposition

Photolithography and Wet Etching for Via

Photolithography and Evaporator for Au/Cr Hall Pads

Photolithography and Evaporation for Writing Line and Pads

Photolithography and Dry Etching for Magnetic Strip

Fig.4.3.9 Fabrication steps of the spintronic convolution engine prototype. (a-b) on a substrate, (c) the magnetic films are deposited. Afterwards, (d) the Hall electrodes are placed on top of the stack, (e) the magnetic strip pattern is defined and SiO2 insulation is deposited. (f) The stack is etched until the electrodes are exposed, which are (g) connected to write line and pads.

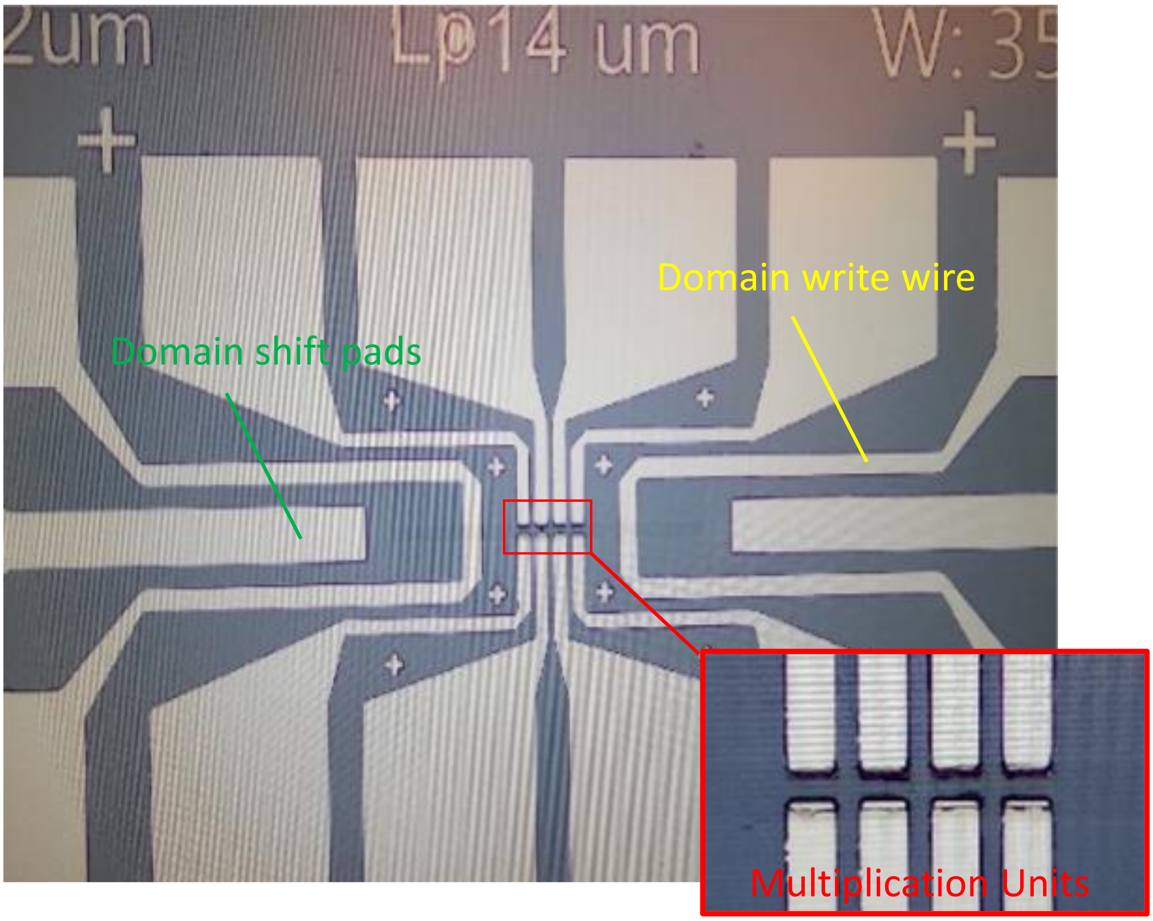


Fig.4.3.10 Completed convolution engine prototype, which include the core composing series of multiplication units, the current path for shifting domains, and the write wire to write the domains at the start of the device.

4.4 Magnetic Tunneling Junction-Domain Wall Motion (MTJ-DWM) Convolution Engine

4.4a Device Structure

The Hall-domain convolution engine provides highly efficient, compact signal processing. However, it is limited to applications with predetermined coefficients such as filtering and transforms. This is because one input of the convolution engine maps to the physical distance between Hall electrodes, a parameter that is determined at fabrication time. After fabrication, the distance between the Hall electrodes cannot be modified. However, in generalized machine learning applications, it is desired that all signals in the convolution engine can be programmed.

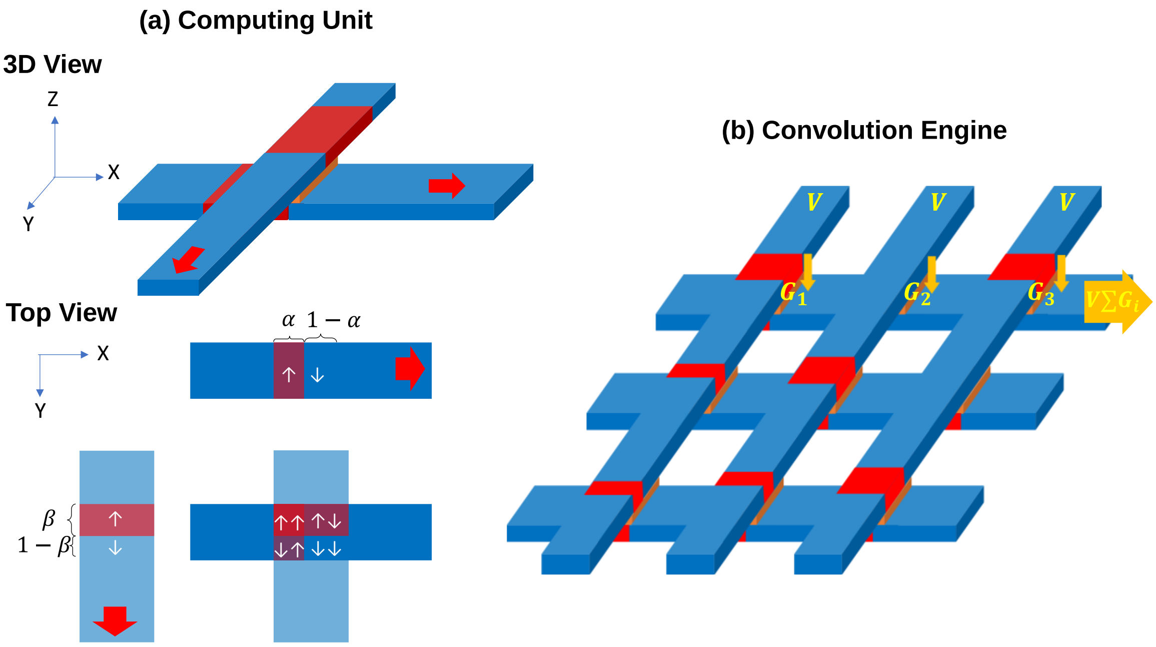


Fig.4.4.1 MTJ-domain based convolution engine. (a) the multiplication unit with the structure of an MTJ formed at the intersection of two magnetic strips. The conductance of the MTJ is proportional to the product of the magnetization of the two strips at the junction. (b) the convolution engine composes an array of multiplication units. During each cycle, the conductance of MTJs on each strip are summed via KCL; after which domains are shifted to the adjacent multiplication unit.

Our MTJ-domain approach overcomes this limitation by utilizing a crossbar array of domain-hosting magnetic strips, all of which can be programmed at computation time. At each cross-point in the array, an MTJ is formed. The tunneling magnetoresistance effect is used for multiplication and readout, and summation is achieved using Kirchhoff's current law (KCL). In addition, signals moving in both the vertical and horizontal strips provides flexible dataflow which further enables novel architectures and multi-dimensional convolutions.

Multiplication Unit

The multiplication unit is shown in **Fig.4.4.1a**, built via a tunneling barrier or a full MTJ placed at the intersection of two domain-hosting magnetic strips. The square-shaped MTJ has dimensions w_{TJ}. At the junction, the x −direction strip has w_α of its area pointing in the $+z$ direction and the remaining $w_{TJ} - w_\alpha$ length pointing in the $-z$ direction. Likewise, the y −direction strip has w_β of its area pointing in the $+z$ magnetization and the remaining $w_{TJ} - w_\beta$ pointing in the $-z$ direction. The MTJ conductance has 4 components: (1) the overlap between w_α and w_β in the parallel $(+z, +z)$ state; (2) the overlap of w_α and $w_{TJ} - w_\beta$ region in the anti-parallel $(+z, -z)$ state; (3) the overlap of $w_{TJ} - w_\alpha$ and w_β in the anti-parallel $(-z, +z)$ state, and (4) the overlap of $w_{TJ} - w_\alpha$ and $w_{TJ} - w_\beta$ in the parallel state $(-z, -z)$. The total conductance is therefore:

$$G_{TJ} = G_P\left(w_\alpha w_\beta + (w_{TJ} - w_\alpha)(w_{TJ} - w_\beta)\right)$$
$$+G_{AP}\left(w_\alpha(w_{TJ} - w_\beta) + (w_{TJ} - w_\alpha)w_\beta\right) \quad \ldots Eq. 4.4.1$$

Where $G_{AP} = R_{AP}^{-1} = TG_P$ is the AP state unit conductance, G_P is the P state unit conductance, and $T = (1 + TMR)^{-1}$ is the conductance ratio between the two states. Eq.4.4.1 can be rewritten in terms of the net magnetization of the strips at the intersection, e.g. the sum of the contribution from the up and down regions:

$$M_x = C_1(w_\alpha - (w_{TJ} - w_\alpha)) \quad \ldots Eq. 4.4.2$$

$$M_y = C_1(w_\beta - (w_{TJ} - w_\beta)) \quad \ldots Eq. 4.4.3$$

Where C_1 compose of material and device constants. Plugging Eq.4.4.2 and Eq.4.4.3 into Eq.4.4.1, we can rewrite the junction conductance as

$$G_{TJ} = C_2(T - 1)G_{AP}M_xM_y + C_2(T + 1)G_{AP}w_{TJ}^2 \quad \ldots Eq. 4.4.4$$

As T, G_{AP}, w_{TJ}^2, and C_2 are all material and design-time constants, Eq.4.4.4 provides the desired multiplication result: the MTJ conductance is the linear product of the net magnetization of the $x-$direction strip at the junction M_x and that of the y−direction strip M_y.

Convolution Engine

The convolution engine (**Fig.4.4.1b**) composes N vertical strips running in the y−direction and M horizontal strips running in the $x-$direction, forming an array of $N \times M$ multiplication units. During the summation process, KCL is used to sum the results of each multiplication unit. This is achieved by clamping the top vertical strips to V_{Read} and the bottom to ground.

The current flowing through the j^{th} bottom strip $I_x[j]$ is the sum of the current flowing through each MTJ on the strip, i.e.

$$I_x[j] = \sum_{k=1}^{N} C_3 + C_4 M_x[j,k] M_y[j,k] \quad \ldots Eq. 4.4.5$$

Where C_3 and C_4 are composed of design and material constants and V_{Read}. For each strip (e.g. fixed j), shifting signals in the $x-$direction results in a convolution:

$$I_x[n] = \sum_{k=1}^{N} C_3 + C_4 M_x[k-n] M_y[k] \quad \ldots Eq. 4.4.6$$

4.4b Performance Analysis

Device dimensions

The minimum dimensions of each computing unit follow the restrictions as Section 4.3b, i.e. (1) The domain should be stable and reach target thermal stability (Eq.4.3.7-4.3.8), and (2) it covers the input resolution (Eq.4.3.9). This results in the dimensions $w_{TJ} = 28nm$ for a binary multiplication unit and $w_{TJ} = 53nm$ for an 8-bit multiplication unit. Note that, the dimensions for the MTJ-DWM design are smaller than the AHE-DWM design, as there is no need for electrodes to be placed upon each multiplication unit.

Operation Frequency

Compared to the Hall-domain convolution engine, the MTJ approach requires no sampling during the summation process. Instead, the speed is limited by the total RC delay along the strip. This can be modeled as an $N-$stage RC ladder, with the speed capped approximately by the delay of the worst-case cell. This cell has an RC constant of

$$\left(R_{TJ} + 2R_{strip}\right)C_{unit} \quad \ldots Eq. 4.4.7$$

Where R_{TJ} is the MTJ resistance, R_{strip} is the resistance of the strip at the worst-case unit, and C_{unit} is the capacitance of each computing unit. This results in a delay of 630ps for a binary 1024-point convolution engine and 1.7ns for an 8-bit 1024-point convolution.

Table 4.4.1 Performance of the 1024-point MTJ-domain convolution engine.

Quantity	**MTJ-domain**	
	Binary	8Bit
Area (A)	$3.2\ \mu\mathrm{m}^2$	$6.6\ \mu\mathrm{m}^2$
Throughput (T)	183 MOp/s	128 MOp/s
Energy (E)	8.4×10^{-13}J	6.3×10^{-12}J
FOM = T/EA	1.4×10^{32}	6.1×10^{30}

During the write and shift phase, the speed is limited by the time necessary to drive domains from one computing unit to the next $(w_{TJ} + F)/\ \delta$. With the dimensions computed above, we can obtain the delay of a binary convolution engine to be 2.7ns (183MHz), and that of an 8-bit engine is 3.9ns (128MHz).

Energy Consumption

Similar to the AHE-DWM processor, the MTJ-DWM processor has a shift and input phase and a summation phase. During the summation phase, the energy consumption includes: (1) the read current through the magnetic strip, (2) the read current through each MTJ device, and (3) the charging of the magnetic strip and MTJ capacitance. Due to the presence of an insulating barrier, the resistance and capacitance of the MTJ is significantly larger than that of the strip (about 10~100x). Thus, the energy consumption can be approximated as:

$$E_{Read} = C_{MTJ}V_{Read}^2 + V_{Read}^2/R_{MTJ}t_1 + \ I_{Read}^2 R_{Strip}t_1 \qquad \dots Eq. 4.4.8$$

Where C_{MTJ} and R_{MTJ} are the capacitance and resistance of the MTJ, V_{Read} is the read voltage, I_{Read} is the read current (which is the sum of all MTJ currents on the strip), and t_1 is the period of the summation phase. Using experimental values of 1fF and 200k Ω for a 50nm MTJ and a read voltage of 0.2V, this energy is $5.5 \times 10^{-16} J$ for binary convolution and $5.1 \times 10^{-15} J$ for an 8-bit convolution.

During the shift and input phase, the energy consumption follows that Eq.4.3.12; e.g. the domain shift energy and the domain generation energy. This results in an energy of $2.7 \times 10^{-16} J$ for binary convolution and $3.9 \times 10^{-15} J$ for an 8-bit convolution.

From the above analysis, we obtain an energy, area, and operation frequency of $3.2\mu m^2, 841 fJ, and\ 183MHz$ for the binary 1024-point convolution engine, and $6.6\mu m^2$, $6.3pJ,\ and\ 128MHz$ for the 8-bit convolution engine. A summary of the two spintronic approaches is shown in **Table 4.4.1**. Compared to the Hall-based approach, the MTJ-based convolution engine provides improved performance, area, and scalability. The is due to (1) a simpler multiplication unit structure (without electrodes), and (2) removal of sampling process and circuitry.

4.4c Simulation

While the MTJ-domain approach outperforms the Hall-domain approach, the fabrication of multiple layers of magnetic patterns (e.g. strip-barrier-strip or strip-MTJ-strip) is more challenging than placing electrodes on a magnetic strip. There is yet to be mature technology for such a process. Hence, we verify the functionality of the MTJ-domain approach via simulation.

We compose an array that has 20 vertical strips, 20 horizontal strips, and a total of 400 MTJs. We set the dimension of each MTJ and their separation to be 10 units. An additional 40 units of magnetic strip is added outside of the array for signal input.

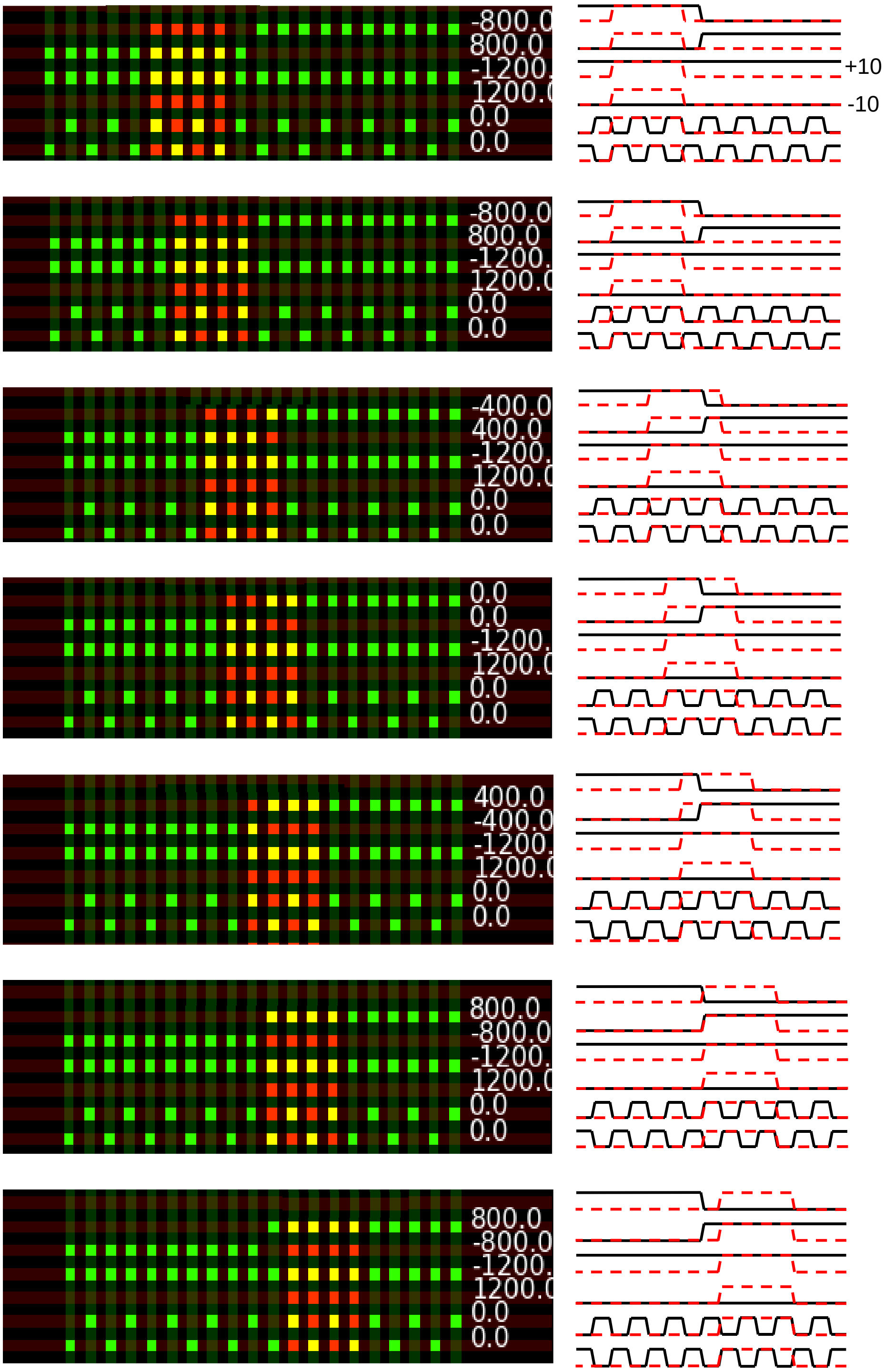


Fig.4.4.2 Simulation of the MTJ-domain convolution engine. In each image, we show the vertical/horizonal strips in red/green and their domains in bright red/green. Overlaps are shown in yellow. The patterns from top to bottom: positive step, negative step, constant max, constant min, interleaved max/min, and interleaved min/max. These patterns are convolved with a square wave. The convolution output is shown at the end of each horizontal strip. To the right of each image, the corresponding signals for each strip is displayed.

The spin orientation in each unit is represented as +1 or -1. The maximum signal, $w_{\alpha} = 10$, corresponds to a numerical value of $+10$, and the minimum signal $w_\alpha = 0$ to a numerical value of -10. Each step in w_α corresponds to a change of 2 in the numerical value

Fig.4.4.2 shows the simulation of the engine in computing the convolution of a square wave with the following patterns: (1)a positive step, (2)a negative step, (3)constant max-values, (4) constant min-values, (5)interleaved max-min patterns, and (5)interleaved min-max patterns. Vertical (horizontal) strips are displayed in green (red) and its +z magnetization domains in bright green (red). Overlaps of the strips and domains are displayed in yellow/bright yellow. The computation result is displayed at the end of each strip.

On the top strip, the convolution of the square pulse with a positive step is initially at -800 when there is no overlap between the pulse and the step. As the amount of overlap increases, the output signal gradually increases and caps at 800. The inverse case is observed for the negative step on the second strip, where the signal is initially at 800 and gradually decreases to -800 as the overlap amount reduces. For constant max and min values, the output is always -1200 and 1200; and for interleaved max/mins the output is always 0. These results confirm the operation of the MTJ-based convolution engine.

CHAPTER 5

Conclusion and Perspectives

Neural networks and AI are becoming increasingly involved in everyday life. Advancing the technology involves innovation across the entire computing hierarchy, from the algorithm, hardware, down to the individual physical devices. New network structures, optimization methods, specialized hardware accelerators, and emerging devices all contribute to the ever-expanding frontier of AI. In this dissertation, we present our perspectives towards each computing hierarchy.

At the algorithm level, we take inspiration from special neurons in biological information processing systems to develop application-specific neural networks. In chapter 2, we first describe the behavior and response of audio and visual neurons and propose a method to compose their neuron models. Through neural architecture search (NAS), we optimize networks containing these neurons through three complementary NAS methods, each of which uses a different bio-inspired search mechanism: mutation, growth, and pruning. NAS is conducted on a hierarchy of datasets that include visual recognition on pictures, handwritten characters, and artwork; as well as audio recognition on speech, environmental sounds, and music. We then analyze the characteristics of high-performing networks to develop specialized networks and demonstrate their application on state-of-the-art networks.

At the hardware level, we recognize the importance of memory in AI hardware and the challenges brought forth by memory scaling. We propose two schemes to overcome limitations caused by variation, based on compensation of variation and variation-tolerant designs. In chapter 3, after an introduction on the memory circuit architecture and variation sources, we describe our 2D calibration scheme, which composes a calibration grid to cancel out variations in fabrication and the access path efficiently. Next, we propose the dual-dataline read scheme, which recycles sensing current to improve read margin, speed, and energy to offset the heavy read performance degradation caused by variation.

Last but not least, utilizing physical mechanisms to create AI-computing devices provides the greatest boost in computing efficiency. In chapter 4, we propose two approaches to compose a spintronic processing engine, one through coupling the Hall effect with domain motion, and the other through coupling the tunneling magnetoresistance effect with domain motion. The former provides a simple, easy-to-fabricate process; while the latter provides higher flexibility and performance. We theoretically analyze the metrics of these devices and show their superior performance over state-of-the-art CMOS technology. Finally, we experimentally verify the mechanisms necessary for the Hall-domain spintronic processor and simulate the operation of the MTJ-domain processor.

The approaches proposed in this dissertation present our perspective towards next-generation AI. We stress that the advancement in AI involves innovation on the entire computing stack; the cooperation, inspiration, and integration of novel neuroscience, physics, designs, and algorithms are equally crucial to overcome the bottlenecks in computation today (**Fig.5.1.1**). This dissertation displays our efforts towards tackling the challenges in three computing hierarchies. Here, we further identify several future paths based on the proposed works. Towards our algorithmic perspective, the involvement of additional neuron types, neural behavior, and network structures beyond what was discussed in this dissertation could improve AI performance and provide insight into how human-like information processing could be achieved. Additionally, new sensing circuits that take advantage of the dual-data line read scheme, and robust systems that expand 2D calibration to N-Dimensional or machine-learning-based calibrations on memory cubes may be enhanced and continue to expand the memory efficiency. Materials like antiferromagnets and magnetic insulators may provide

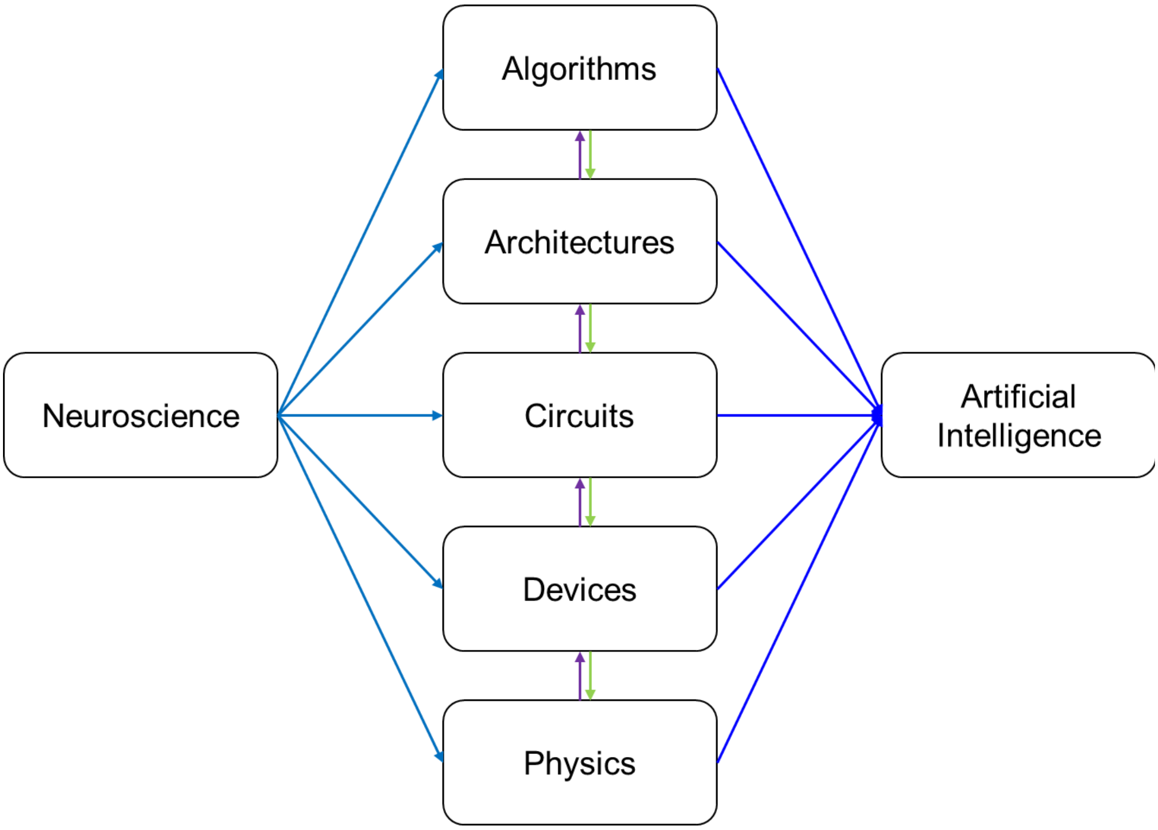


Fig.5.1.1 The path towards next generation AI: inspiration from neuroscience could lead to innovation across different layers of the computing hierarchy. Advancing AI involves parallel development of ideas in physics, devices, circuits, system architectures, and algorithms; as well as vertical integration involving multiple layers of the hierarchy.

improved scalability, faster speeds (domain speeds can reach up to thousands of meters per second), and potentially remove the summation circuitry in the convolution engine (with insulating materials, AHE voltages can be directly connected in series rather than require sampling); while improving the summation circuitry and analog-digital converters can interface with digital components. The author holds an optimistic prospect about the future of AI.

References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.

[2] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.

[3] D. O. Hebb, *The Organization of Behavior*. New York: Wiley & Sons, 1949.

[4] A. G. Ivakhnenko, "The group method of data of handling; a rival of the method of stochastic approximation," *Sov. Autom. Control*, vol. 13, pp. 43–55, 1968.

[5] N. R. C. (US). A. L. P. A. Committee and A. (Organization), *Language and Machines: Computers in Translation and Linguistics; a Report*, vol. 1416. National Academies, 1966.

[6] J. Lighthill, "Artificial intelligence: A general survey," in *Artificial Intelligence: a paper symposium*, 1973, pp. 1–21.

[7] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[8] "AI winter - Wikipedia." https://en.wikipedia.org/wiki/AI_winter (accessed Mar. 08, 2021).

[9] P. J. Werbos, *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, vol. 1. John Wiley & Sons, 1994.

[10] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, 1980.

[11] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci.*, vol. 79, no. 8, pp. 2554–2558, 1982.

[12] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.

[13] M. Campbell, A. J. Hoane Jr, and F. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, no. 1–2, pp. 57–83, 2002.

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

[15] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255, doi: 10.1109/CVPRW.2009.5206848.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012, doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007.

[17] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[18] N. NVIDIA, "GP100 Pascal Whitepaper," 2016.

[19] I. Buck, “Gpu computing with nvidia cuda,” in *ACM SIGGRAPH 2007 courses*, 2007, pp. 6-es.

[20] M. Abadi *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[21] R. Al-Rfou *et al.*, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, p. arXiv-1605, 2016.

[22] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[23] J. Ouyang, X. Du, Y. Ma, and J. Liu, “Kunlun: A 14nm High-Performance AI Processor for Diversified Workloads,” in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, 2021, vol. 64, pp. 50–51.

[24] K. Ueyoshi *et al.*, “QUEST: A 7.49 TOPS multi-purpose log-quantized DNN inference engine stacked on 96MB 3D SRAM using inductive-coupling technology in 40nm CMOS,” in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2018, pp. 216–218.

[25] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, “UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision,” in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2018, pp. 218–220.

[26] P. C. Knag *et al.*, “A 617-TOPS/W All-Digital Binary Neural Network Accelerator in 10-nm FinFET CMOS,” *IEEE J. Solid-State Circuits*, 2020.

[27] K. Ando *et al.*, “BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W,” *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, 2017.

[28] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, “Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices,” *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 9, no. 2, pp. 292–308, 2019.

[29] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, “YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights,” in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 236–241.

[30] A. Amravati, S. Bin Nasir, S. Thangadurai, I. Yoon, and A. Raychowdhury, “A 55nm time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots,” in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2018, pp. 124–126.

[31] Z. Chen, S. Fu, Q. Cao, and J. Gu, “A Mixed-Signal Time-Domain Generative Adversarial Network Accelerator with Efficient Subthreshold Time Multiplier and Mixed-Signal On-Chip Training for Low Power Edge Devices,” in *2020 IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.

[32] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, “DeltaRNN: A power-efficient recurrent neural network accelerator,” in *Proceedings of the 2018 ACM/SIGDA*

International Symposium on Field-Programmable Gate Arrays, 2018, pp. 21–30.

[33] P. Chi *et al.*, “Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory,” in *ACM SIGARCH Computer Architecture News*, 2016, vol. 44, no. 3, pp. 27–39.

[34] W.-H. Chen *et al.*, “A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors,” in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2018, pp. 494–496.

[35] C.-X. Xue *et al.*, “24.1 A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors,” in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2019, pp. 388–390.

[36] W.-H. Chen *et al.*, “CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors,” *Nat. Electron.*, vol. 2, no. 9, pp. 420–428, 2019.

[37] G. W. Burr *et al.*, “Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element,” *IEEE Trans. Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

[38] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A survey of the recent architectures of deep convolutional neural networks,” *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2020.

[39] W. Pauli, “The connection between spin and statistics,” *Phys. Rev.*, vol. 58, no. 8, p. 716, 1940.

[40] L. Landau and E. Lifshitz, “On the theory of the dispersion of magnetic permeability in ferromagnetic bodies,” in *Perspectives in Theoretical Physics*, Elsevier, 1992, pp. 51–65.

[41] M. I. D’Yakonov and V. I. Perel, “Possibility of orienting electron spins with current,” *Sov. J. Exp. Theor. Phys. Lett.*, vol. 13, p. 467, 1971.

[42] R. Meservey, D. Paraskevopoulos, and P. M. Tedrow, “Correlation between Spin Polarization of Tunnel Currents from 3 d Ferromagnets and Their Magnetic Moments,” *Phys. Rev. Lett.*, vol. 37, no. 13, p. 858, 1976.

[43] M. Julliere, “Tunneling between ferromagnetic films,” *Phys. Lett. A*, vol. 54, no. 3, pp. 225–226, 1975.

[44] M. N. Baibich *et al.*, “Giant magnetoresistance of (001) Fe/(001) Cr magnetic superlattices,” *Phys. Rev. Lett.*, vol. 61, no. 21, p. 2472, 1988.

[45] E. B. Myers, D. C. Ralph, J. A. Katine, R. N. Louie, and R. A. Buhrman, “Current-induced switching of domains in magnetic multilayer devices,” *Science (80-.).*, vol. 285, no. 5429, pp. 867–870, 1999.

[46] S. Zhang and Z. Li, “Roles of nonequilibrium conduction electrons on the magnetization dynamics of ferromagnets,” *Phys. Rev. Lett.*, vol. 93, no. 12, p. 127204, 2004.

[47] T. Maruyama *et al.*, “Large voltage-induced magnetic anisotropy change in a few atomic layers of iron,” *Nat. Nanotechnol.*, vol. 4, no. 3, pp. 158–161, 2009.

[48] P. K. Amiri and K. L. Wang, “Voltage-controlled magnetic anisotropy in spintronic devices,” in *Spin*, 2012, vol. 2, no. 03, p. 1240002.

[49] X. Li *et al.*, “Enhancement of voltage-controlled magnetic anisotropy through precise control of Mg insertion thickness at CoFeB| MgO interface,” *Appl. Phys. Lett.*, vol. 110, no. 5, p. 52401, 2017.

[50] L. Liu, C.-F. Pai, Y. Li, H. W. Tseng, D. C. Ralph, and R. A. Buhrman, “Spin-Torque Switching with the Giant Spin Hall Effect of Tantalum,” *Science (80-.).*, vol. 336, no. 6081, pp. 555–558, 2012, doi: 10.1126/science.1218197.

[51] G. Yu *et al.*, “Switching of perpendicular magnetization by spin–orbit torques in the absence of external magnetic fields,” *Nat. Nanotechnol.*, vol. 9, no. 7, pp. 548–554, 2014, doi: 10.1038/nnano.2014.94.

[52] I. M. Miron *et al.*, “Perpendicular switching of a single ferromagnetic layer induced by in-plane current injection,” *Nature*, vol. 476, no. 7359, pp. 189–193, 2011, doi: 10.1038/nature10309.

[53] N. Nagaosa and Y. Tokura, “Topological properties and dynamics of magnetic skyrmions,” *Nat. Nanotechnol.*, vol. 8, no. 12, pp. 899–911, 2013.

[54] A. Fert, N. Reyren, and V. Cros, “Magnetic skyrmions: advances in physics and potential applications,” *Nat. Rev. Mater.*, vol. 2, no. 7, pp. 1–15, 2017.

[55] W. Jiang *et al.*, “Blowing magnetic skyrmion bubbles,” *Science (80-.).*, vol. 349, no. 6245, pp. 283–286, 2015.

[56] S. Sachdev and J. Ye, “Gapless spin-fluid ground state in a random quantum Heisenberg magnet,” *Phys. Rev. Lett.*, vol. 70, no. 21, p. 3339, 1993.

[57] G. Shirane *et al.*, “Two-dimensional antiferromagnetic quantum spin-fluid state in La 2 CuO 4,” *Phys. Rev. Lett.*, vol. 59, no. 14, p. 1613, 1987.

[58] F. Lado and E. Lomba, “Heisenberg spin fluid in an external magnetic field,” *Phys. Rev. Lett.*, vol. 80, no. 16, p. 3535, 1998.

[59] D. Sherrington and S. Kirkpatrick, “Solvable model of a spin-glass,” *Phys. Rev. Lett.*, vol. 35, no. 26, p. 1792, 1975.

[60] M. Ali, P. Adie, C. H. Marrows, D. Greig, B. J. Hickey, and R. L. Stamps, “Exchange bias using a spin glass,” *Nat. Mater.*, vol. 6, no. 1, pp. 70–75, 2007.

[61] G. E. Santoro, R. Martoňák, E. Tosatti, and R. Car, “Theory of quantum annealing of an Ising spin glass,” *Science (80-.).*, vol. 295, no. 5564, pp. 2427–2430, 2002.

[62] S. S. P. Parkin, M. Hayashi, and L. Thomas, “Magnetic domain-wall racetrack memory,” *Science (80-.).*, vol. 320, no. 5873, pp. 190–194, 2008.

[63] D. A. Allwood, G. Xiong, C. C. Faulkner, D. Atkinson, D. Petit, and R. P. Cowburn, “Magnetic domain-wall logic,” *Science (80-.).*, vol. 309, no. 5741, pp. 1688–1692, 2005.

[64] G. Bochi *et al.*, “Magnetic domain structure in ultrathin films,” *Phys. Rev. Lett.*, vol. 75, no. 9, p. 1839, 1995.

[65] I. Dzyaloshinsky, “A thermodynamic theory of ‘weak’ ferromagnetism of

antiferromagnetics," *J. Phys. Chem. Solids*, vol. 4, no. 4, pp. 241–255, 1958.

[66] T. Moriya, "Anisotropic superexchange interaction and weak ferromagnetism," *Phys. Rev.*, vol. 120, no. 1, p. 91, 1960.

[67] A. Fert and P. M. Levy, "Role of anisotropic exchange interactions in determining the properties of spin-glasses," *Phys. Rev. Lett.*, vol. 44, no. 23, p. 1538, 1980.

[68] D. D. Stancil and A. Prabhakar, "' Spin Waves: Theory and Applications', Springer," *New York*, 2009.

[69] A. Spinelli, B. Bryant, F. Delgado, J. Fernández-Rossier, and A. F. Otte, "Imaging of spin waves in atomically designed nanomagnets," *Nat. Mater.*, vol. 13, no. 8, pp. 782–785, 2014.

[70] T. Kampfrath *et al.*, "Coherent terahertz control of antiferromagnetic spin waves," *Nat. Photonics*, vol. 5, no. 1, pp. 31–34, 2011.

[71] K. Uchida *et al.*, "Observation of the spin Seebeck effect," *Nature*, vol. 455, no. 7214, pp. 778–781, 2008.

[72] K. Uchida *et al.*, "Spin seebeck insulator," *Nat. Mater.*, vol. 9, no. 11, pp. 894–897, 2010.

[73] H. Adachi, K. Uchida, E. Saitoh, and S. Maekawa, "Theory of the spin Seebeck effect," *Reports Prog. Phys.*, vol. 76, no. 3, p. 36501, 2013.

[74] Z. Q. Qiu and S. D. Bader, "Surface magneto-optic Kerr effect," *Rev. Sci. Instrum.*, vol. 71, no. 3, pp. 1243–1255, 2000.

[75] Z. Q. Qiu and S. D. Bader, "Surface magneto-optic Kerr effect (SMOKE)," *J. Magn. Magn. Mater.*, vol. 200, no. 1–3, pp. 664–678, 1999.

[76] G. P. Zhang, W. Hübner, G. Lefkidis, Y. Bai, and T. F. George, "Paradigm of the time-resolved magneto-optical Kerr effect for femtosecond magnetism," *Nat. Phys.*, vol. 5, no. 7, pp. 499–502, 2009.

[77] S. Yuasa, A. Fukushima, H. Kubota, Y. Suzuki, and K. Ando, "Giant tunneling magnetoresistance up to 410% at room temperature in fully epitaxial Co⁄Mg O⁄Co magnetic tunnel junctions with bcc Co (001) electrodes," *Appl. Phys. Lett.*, vol. 89, no. 4, p. 42505, 2006.

[78] S. Yuasa and D. D. Djayaprawira, "Giant tunnel magnetoresistance in magnetic tunnel junctions with a crystalline MgO (0 0 1) barrier," *J. Phys. D. Appl. Phys.*, vol. 40, no. 21, p. R337, 2007.

[79] S. Ikeda *et al.*, "Tunnel magnetoresistance of 604% at 300 K by suppression of Ta diffusion in Co Fe B⁄Mg O⁄Co Fe B pseudo-spin-valves annealed at high temperature," *Appl. Phys. Lett.*, vol. 93, no. 8, p. 82508, 2008.

[80] Q. Dong *et al.*, "A 1Mb 28nm STT-MRAM with 2.8 ns read access time at 1.2 V VDD using single-cap offset-cancelled sense amplifier and in-situ self-write-termination," in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, 2018, pp. 480–482.

[81] T.-H. Yang, K.-X. Li, Y.-N. Chiang, W.-Y. Lin, H.-T. Lin, and M.-F. Chang, "A 28nm

32Kb embedded 2T2MTJ STT-MRAM macro with 1.3 ns read-access time for fast and reliable read applications," in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, 2018, pp. 482–484.

[82] K. Rho *et al.*, "23.5 A 4Gb LPDDR2 STT-MRAM with compact 9F2 1T1MTJ cell and hierarchical bitline architecture," in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, 2017, pp. 396–397.

[83] Y. C. Wu *et al.*, "Deterministic and field-free voltage-controlled MRAM for high performance and low power applications," in *2020 IEEE Symposium on VLSI Technology*, 2020, pp. 1–2.

[84] C. Grezes *et al.*, "Write error rate and read disturbance in electric-field-controlled magnetic random-access memory," *IEEE Magn. Lett.*, vol. 8, pp. 1–5, 2016.

[85] P. K. Amiri *et al.*, "Electric-field-controlled magnetoelectric RAM: progress, challenges, and scaling," *IEEE Trans. Magn.*, vol. 51, no. 11, pp. 1–7, 2015.

[86] K. Garello *et al.*, "Manufacturable 300mm platform solution for field-free switching SOT-MRAM," in *2019 Symposium on VLSI Circuits*, 2019, pp. T194–T195.

[87] K. Garello *et al.*, "SOT-MRAM 300mm integration for low power and ultrafast embedded memories," in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 81–82.

[88] S. Bhanja, D. K. Karunaratne, R. Panchumarthy, S. Rajaram, and S. Sarkar, "Non-Boolean computing with nanomagnets for computer vision applications," *Nat. Nanotechnol.*, vol. 11, no. 2, p. 177, 2016.

[89] B. Sutton, K. Y. Camsari, B. Behin-Aein, and S. Datta, "Intrinsic optimization using stochastic nanomagnets," *Sci. Rep.*, vol. 7, p. 44370, 2017.

[90] P. Debashis, R. Faria, K. Y. Camsari, J. Appenzeller, S. Datta, and Z. Chen, "Experimental demonstration of nanomagnet networks as hardware for Ising computing," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 33–34.

[91] H. Goto, "Bifurcation-based adiabatic quantum computation with a nonlinear oscillator network," *Sci. Rep.*, vol. 6, no. 1, pp. 1–8, 2016.

[92] F. Macià, A. D. Kent, and F. C. Hoppensteadt, "Spin-wave interference patterns created by spin-torque nano-oscillators for memory and computation," *Nanotechnology*, vol. 22, no. 9, p. 95301, 2011.

[93] H. Arai and H. Imamura, "Neural-network computation using spin-wave-coupled spin-torque oscillators," *Phys. Rev. Appl.*, vol. 10, no. 2, p. 24040, 2018.

[94] J. Torrejon *et al.*, "Neuromorphic computing with nanoscale spintronic oscillators," *Nature*, vol. 547, no. 7664, p. 428, 2017.

[95] M. Romera *et al.*, "Vowel recognition with four coupled spin-torque nano-oscillators," *Nature*, vol. 563, no. 7730, pp. 230–234, 2018.

[96] S. P. Levitan, Y. Fang, D. H. Dash, T. Shibata, D. E. Nikonov, and G. I. Bourianoff, "Non-Boolean associative architectures based on nano-oscillators," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, 2012, pp. 1–6.

[97] H. Nomura *et al.*, “Reservoir computing with dipole-coupled nanomagnets,” *Jpn. J. Appl. Phys.*, vol. 58, no. 7, p. 70901, 2019.

[98] D. Pinna, G. Bourianoff, and K. Everschor-Sitte, “Reservoir computing with random skyrmion textures,” *Phys. Rev. Appl.*, vol. 14, no. 5, p. 54020, 2020.

[99] G. Csaba, A. Papp, W. Porod, and R. Yeniceri, “Non-boolean computing based on linear waves and oscillators,” in *2015 45th European Solid State Device Research Conference (ESSDERC)*, 2015, pp. 101–104.

[100] G. Csaba, Á. Papp, and W. Porod, “Perspectives of using spin waves for computing and signal processing,” *Phys. Lett. A*, vol. 381, no. 17, pp. 1471–1476, 2017.

[101] A. Mizrahi *et al.*, “Neural-like computing with populations of superparamagnetic basis functions,” *Nat. Commun.*, vol. 9, no. 1, pp. 1–11, 2018.

[102] W. A. Borders *et al.*, “Analogue spin–orbit torque device for artificial-neural-network-based associative memory operation,” *Appl. Phys. express*, vol. 10, no. 1, p. 13007, 2016.

[103] D. Prychynenko *et al.*, “Magnetic skyrmion as a nonlinear resistive element: a potential building block for reservoir computing,” *Phys. Rev. Appl.*, vol. 9, no. 1, p. 14034, 2018.

[104] D. Fan, Y. Shim, A. Raghunathan, and K. Roy, “STT-SNN: A spin-transfer-torque based soft-limiting non-linear neuron for low-power artificial neural networks,” *IEEE Trans. Nanotechnol.*, vol. 14, no. 6, pp. 1013–1023, 2015.

[105] S. R. Kulkarni, D. V. Kadetotad, S. Yin, J.-S. Seo, and B. Rajendran, “Neuromorphic hardware accelerator for snn inference based on stt-ram crossbar arrays,” in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 438–441.

[106] Y. Huang, W. Kang, X. Zhang, Y. Zhou, and W. Zhao, “Magnetic skyrmion-based synaptic devices,” *Nanotechnology*, vol. 28, no. 8, p. 08LT02, Jan. 2017, doi: 10.1088/1361-6528/aa5838.

[107] K. Y. Camsari, S. Salahuddin, and S. Datta, “Implementing p-bits with embedded MTJ,” *IEEE Electron Device Lett.*, vol. 38, no. 12, pp. 1767–1770, 2017.

[108] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, “Stochastic p-bits for invertible logic,” *Phys. Rev. X*, vol. 7, no. 3, p. 31014, 2017.

[109] A. Lee, B. Dai, D. Wu, H. Wu, R. N. Schwartz, and K. L. Wang, “A thermodynamic core using voltage-controlled spin-orbit-torque magnetic tunnel junctions.,” *Nanotechnology*, Mar. 2021, doi: 10.1088/1361-6528/abeb9b.

[110] H. Zeng and J. R. Sanes, “Neuronal cell-type classification: challenges, opportunities and the path forward,” *Nat. Rev. Neurosci.*, vol. 18, no. 9, pp. 530–546, 2017.

[111] J. O’Keefe, N. Burgess, J. G. Donnett, K. J. Jeffery, and E. A. Maguire, “Place cells, navigational accuracy, and the human hippocampus,” *Philos. Trans. R. Soc. London. Ser. B Biol. Sci.*, vol. 353, no. 1373, pp. 1333–1340, 1998.

[112] R. Muller, “A quarter of a century of place cells,” *Neuron*, vol. 17, no. 5, pp. 813–822, 1996.

[113] E. I. Moser, E. Kropff, and M.-B. Moser, “Place Cells, Grid Cells, and the Brain’s Spatial Representation System,” *Annu. Rev. Neurosci.*, vol. 31, no. 1, pp. 69–89, 2008, doi: 10.1146/annurev.neuro.31.061307.090723.

[114] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser, “Microstructure of a spatial map in the entorhinal cortex,” *Nature*, vol. 436, no. 7052, pp. 801–806, 2005.

[115] J. O’Keefe and M. L. Recce, “Phase relationship between hippocampal place units and the EEG theta rhythm,” *Hippocampus*, vol. 3, no. 3, pp. 317–330, 1993.

[116] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” 2010.

[117] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[118] Y. Sun, X. Wang, and X. Tang, “Deeply learned face representations are sparse, selective, and robust,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2892–2900.

[119] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[120] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, 2013, vol. 30, no. 1, p. 3, [Online]. Available: https://pdfs.semanticscholar.org/367f/2c63a6f6a10b3b64b8729d601e69337ee3cc.pdf.

[121] A. Howard *et al.*, “Searching for mobilenetv3,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[122] T. J. Gawne and J. M. Martin, “Responses of primate visual cortical V4 neurons to simultaneously presented stimuli,” *J. Neurophysiol.*, vol. 88, no. 3, pp. 1128–1135, 2002.

[123] I. Lampl, D. Ferster, T. Poggio, and M. Riesenhuber, “Intracellular measurements of spatial integration and the MAX operation in complex cells of the cat primary visual cortex.,” *J. Neurophysiol.*, vol. 92, no. 5, pp. 2704–2713, 2004, doi: 10.1152/jn.00060.2004.

[124] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, “Robust object recognition with cortex-like mechanisms,” *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 3, pp. 411–426, 2007.

[125] K. Fukushima and S. Miyake, “Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position,” *Pattern Recognit.*, vol. 15, no. 6, pp. 455–469, 1982.

[126] Y.-L. Boureau, J. Ponce, and Y. LeCun, “A theoretical analysis of feature pooling in visual recognition,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 111–118.

[127] E. D. Young and M. B. Sachs, “Representation of steady-state vowels in the temporal aspects of the discharge patterns of populations of auditory-nerve fibers,” *J. Acoust.*

Soc. Am., vol. 66, no. 5, pp. 1381–1403, 1979.

[128] R. V Shannon, F.-G. Zeng, V. Kamath, J. Wygonski, and M. Ekelid, “Speech recognition with primarily temporal cues,” *Science (80-.).*, vol. 270, no. 5234, pp. 303–304, 1995.

[129] N. L. Golding, D. Robertson, and D. Oertel, “Recordings from slices indicate that octopus cells of the cochlear nucleus detect coincident firing of auditory nerve fibers with temporal precision,” *J. Neurosci.*, vol. 15, no. 4, pp. 3138–3153, 1995.

[130] D. Oertel, R. Bal, S. M. Gardner, P. H. Smith, and P. X. Joris, “Detection of synchrony in the activity of auditory nerve fibers by octopus cells of the mammalian cochlear nucleus.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 97, no. 22, pp. 11773–9, 2000, doi: 10.1073/pnas.97.22.11773.

[131] A. Graves, N. Jaitly, and A. Mohamed, “Hybrid speech recognition with deep bidirectional LSTM,” in *2013 IEEE workshop on automatic speech recognition and understanding*, 2013, pp. 273–278.

[132] A. Graves, A. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE international conference on acoustics, speech and signal processing*, 2013, pp. 6645–6649.

[133] Y. Xu, Q. Kong, W. Wang, and M. D. Plumbley, “Large-scale weakly supervised audio classification using gated convolutional neural network,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 121–125.

[134] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey.,” *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[135] B. Zoph, V. Vasudevan, J. Shlens, and Q. V Le, “Learning transferable architectures for scalable image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[136] B. Zoph and Q. V Le, “Neural architecture search with reinforcement learning,” *arXiv Prepr. arXiv1611.01578*, 2016.

[137] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” *arXiv Prepr. arXiv1611.02167*, 2016.

[138] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, “Hierarchical representations for efficient architecture search,” *arXiv Prepr. arXiv1711.00436*, 2017.

[139] E. Real, A. Aggarwal, Y. Huang, and Q. V Le, “Regularized evolution for image classifier architecture search,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, vol. 33, pp. 4780–4789.

[140] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “Smash: one-shot model architecture search through hypernetworks,” *arXiv Prepr. arXiv1708.05344*, 2017.

[141] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *J. Mach. Learn. Res.*, vol. 13, no. Feb, pp. 281–305, 2012.

[142] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture search,” *arXiv Prepr. arXiv1806.09055*, 2018.

[143] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[144] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, “Efficient neural architecture search via parameters sharing,” in *International Conference on Machine Learning*, 2018, pp. 4095–4104.

[145] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, “Efficient architecture search by network transformation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, vol. 32, no. 1.

[146] B. Baker, O. Gupta, R. Raskar, and N. Naik, “Accelerating neural architecture search using performance prediction,” *arXiv Prepr. arXiv1705.10823*, 2017.

[147] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, “Path-level network transformation for efficient architecture search,” in *International Conference on Machine Learning*, 2018, pp. 678–687.

[148] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017, vol. 31, no. 1.

[149] 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*, 2016, pp. 770–778, [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf.

[150] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, “Densenet: Implementing efficient convnet descriptor pyramids,” *arXiv Prepr. arXiv1404.1869*, 2014.

[151] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[152] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” *arXiv Prepr. arXiv1611.06440*, 2016.

[153] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[154] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Citeseer, 2009.

[155] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, “EMNIST: an extension of MNIST to handwritten letters,” *arXiv*, Feb. 2017, Accessed: Nov. 15, 2020. [Online]. Available: http://arxiv.org/abs/1702.05373.

[156] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain, “ICFHR2016 CROHME: Competition on recognition of online handwritten mathematical expressions,” in *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, Jul. 2016, vol. 0, pp. 607–612, doi: 10.1109/ICFHR.2016.0116.

[157] "Best Artworks of All Time | Kaggle." https://www.kaggle.com/ikarus777/best-artworks-of-all-time (accessed Nov. 15, 2020).

[158] B. Saleh and A. Elgammal, "Large-scale Classification of Fine-Art Paintings: Learning The Right Metric on The Right Feature," May 2015, Accessed: Nov. 15, 2020. [Online]. Available: http://arxiv.org/abs/1505.00855.

[159] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv Prepr. arXiv1804.03209*, 2018.

[160] J. Wilkins, P. Seetharaman, A. Wahl, and B. Pardo, "VOCALSET: A SINGING VOICE DATASET," doi: 10.5281/zenodo.1203819.

[161] K. J. Piczak, "ESC: Dataset for environmental sound classification," in *MM 2015 - Proceedings of the 2015 ACM Multimedia Conference*, 2015, pp. 1015–1018, doi: 10.1145/2733373.2806390.

[162] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 1041–1044.

[163] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Trans. Speech Audio Process.*, vol. 10, no. 5, pp. 293–302, Jul. 2002, doi: 10.1109/TSA.2002.800560.

[164] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, "FMA: A Dataset For Music Analysis," Dec. 2016, Accessed: Nov. 16, 2020. [Online]. Available: http://arxiv.org/abs/1612.01840.

[165] J. Hestness *et al.*, "Deep learning scaling is predictable, empirically," *arXiv Prepr. arXiv1712.00409*, 2017.

[166] D. Shum *et al.*, "CMOS-embedded STT-MRAM Arrays in 2x nm Nodes for GP-MCU applications," in *VLSI Technology, 2017 Symposium on*, 2017, pp. T208–T209.

[167] C.-X. Xue *et al.*, "A 22nm 4Mb 8b-Precision ReRAM Computing-in-Memory Macro with 11.91 to 195.7 TOPS/W for Tiny AI Edge Devices," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, 2021, vol. 64, pp. 245–247.

[168] C.-P. Lo *et al.*, "Embedded 2Mb ReRAM macro with 2.6 ns read access time using dynamic-trip-point-mismatch sampling current-mode sense amplifier for IoE applications," in *2017 Symposium on VLSI Circuits*, 2017, pp. C164–C165.

[169] P. Jain *et al.*, "13.2 A 3.6 Mb 10.1 Mb/mm 2 Embedded Non-Volatile ReRAM Macro in 22nm FinFET Technology with Adaptive Forming/Set/Reset Schemes Yielding Down to 0.5 V with Sensing Time of 5ns at 0.7 V," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2019, pp. 212–214.

[170] H. Y. Cheng *et al.*, "Si Incorporation Into AsSeGe Chalcogenides for High Thermal Stability, High Endurance and Extremely Low $\mathrm {V} _ {\mathrm {th}} $ Drift 3D Stackable Cross-point Memory: IBM/Macronix PCRAM Joint Project," in *2020 IEEE Symposium on VLSI Technology*, 2020, pp. 1–2.

[171] H. Y. Cheng *et al.*, "Novel fast-switching and high-data retention phase-change memory based on new Ga-Sb-Ge material," in *International Electron Devices Meeting*

(IEDM), 2016, pp. 3.5.1-3.5.4, doi: 10.1109/IEDM.2015.7409620.

[172] K. Bourzac, “Has intel created a universal memory technology?[news],” *IEEE Spectr.*, vol. 54, no. 5, pp. 9–10, 2017.

[173] S. Yuasa, T. Nagahama, A. Fukushima, Y. Suzuki, and K. Ando, “Giant room-temperature magnetoresistance in single-crystal Fe/MgO/Fe magnetic tunnel junctions,” *Nat. Mater.*, vol. 3, no. 12, p. 868, 2004.

[174] W. Zhang and T. Li, “Characterizing and mitigating the impact of process variations on phase change based memory systems,” in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 2–13.

[175] X. Guan, S. Yu, and H.-S. P. Wong, “On the switching parameter variation of metal-oxide RRAM—Part I: Physical modeling and simulation methodology,” *IEEE Trans. Electron Devices*, vol. 59, no. 4, pp. 1172–1182, 2012.

[176] A. Lee *et al.*, “A ReRAM-based nonvolatile flip-flop with self-write-termination scheme for frequent-OFF fast-wake-up nonvolatile processors,” *IEEE J. Solid-State Circuits*, vol. 52, no. 8, pp. 2194–2207, 2017.

[177] S.-C. Hsu and C.-F. Chien, “Hybrid data mining approach for pattern extraction from wafer bin map to improve yield in semiconductor manufacturing,” *Int. J. Prod. Econ.*, vol. 107, no. 1, pp. 88–103, 2007.

[178] C.-F. Chien, S.-C. Hsu, and Y.-J. Chen, “A system for online detection and classification of wafer bin map defect patterns for manufacturing intelligence,” *Int. J. Prod. Res.*, vol. 51, no. 8, pp. 2324–2338, 2013.

[179] Y.-S. Jeong, S.-J. Kim, and M. K. Jeong, “Automatic identification of defect patterns in semiconductor wafer maps using spatial correlogram and dynamic time warping,” *IEEE Trans. Semicond. Manuf.*, vol. 21, no. 4, pp. 625–637, 2008.

[180] A. Lee, R. Jagannathan, D. Wu, and K. L. Wang, “A 2-D Calibration Scheme for Resistive Nonvolatile Memories,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 28, no. 6, pp. 1371–1377, 2020.

[181] Y.-C. Lai and S.-Y. Huang, “X-calibration: A technique for combating excessive bitline leakage current in nanometer SRAM designs,” *IEEE J. Solid-State Circuits*, vol. 43, no. 9, pp. 1964–1971, 2008.

[182] F. Ren, H. Park, C.-K. K. Yang, and D. Marković, “Reference calibration of body-voltage sensing circuit for high-speed STT-RAMs,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 60, no. 11, pp. 2932–2939, 2013.

[183] Y. Niki *et al.*, “A digitized replica bitline delay technique for random-variation-tolerant timing generation of SRAM sense amplifiers,” *IEEE J. Solid-State Circuits*, vol. 46, no. 11, pp. 2545–2551, 2011.

[184] Z. Lin *et al.*, “A pipeline replica bitline technique for suppressing timing variation of SRAM sense amplifiers in a 28-nm CMOS process,” *IEEE J. Solid-State Circuits*, vol. 52, no. 3, pp. 669–677, 2016.

[185] J. Wu, J. Zhu, Y. Xia, and N. Bai, “A multiple-stage parallel replica-bitline delay addition technique for reducing timing variation of SRAM sense amplifiers,” *IEEE*

Trans. Circuits Syst. II Express Briefs, vol. 61, no. 4, pp. 264–268, 2014.

[186] J. Chang *et al.*, “12.1 a 7nm 256mb sram in high-k metal-gate finfet technology with write-assist circuitry for low-v min applications,” in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 206–207.

[187] D. Kjaer *et al.*, “Fast static field CIPT mapping of unpatterned MRAM film stacks,” *Meas. Sci. Technol.*, vol. 26, no. 4, p. 45602, 2015.

[188] C.-X. Xue, W.-C. Zhao, T.-H. Yang, Y.-J. Chen, H. Yamauchi, and M.-F. Chang, “A 28-nm 320-kb TCAM macro using split-controlled single-load 14T cell and triple-margin voltage sense amplifier,” *IEEE J. Solid-State Circuits*, vol. 54, no. 10, pp. 2743–2753, 2019.

[189] L. Wei *et al.*, “13.3 A 7Mb STT-MRAM in 22FFL FinFET Technology with 4ns Read Sensing Time at 0.9 V Using Write-Verify-Write Scheme and Offset-Cancellation Sensing Technique,” in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2019, pp. 214–216.

[190] C.-C. Lin *et al.*, “A 256b-wordlength ReRAM-based TCAM with 1ns search-time and 14× improvement in wordlength-energyefficiency-density product using 2.5 T1R cell,” in *Internaltional Solid-State Circuits Conference (ISSCC)*, 2016, pp. 136–137.

[191] M. T. Bohr and I. A. Young, “CMOS scaling trends and beyond,” *IEEE Micro*, vol. 37, no. 6, pp. 20–29, 2017.

[192] W.-S. Khwa *et al.*, “A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors,” in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2018, pp. 496–498.

[193] J. J. Kan *et al.*, “Systematic validation of 2x nm diameter perpendicular MTJ arrays and MgO barrier for sub-10 nm embedded STT-MRAM with practically unlimited endurance,” in *International Electron Devices Meeting (IEDM)*, 2017, pp. 27.4.1-27.4.4, doi: 10.1109/IEDM.2016.7838493.

[194] K. L. Wang, J. G. Alzate, and P. Khalili Amiri, “Low-power non-volatile spintronic memory: STT-RAM and beyond,” *J. Phys. D. Appl. Phys.*, vol. 46, no. 7, p. 074003, 2013, doi: 10.1088/0022-3727/46/7/074003.

[195] D. D. Buss, R. W. Brodersen, and C. R. Hewes, “Charge-coupled devices for analog signal processing,” *Proc. IEEE*, vol. 64, no. 5, pp. 801–804, 1976.

[196] R. Brodersen, H.-S. Fu, R. Frye, and D. Buss, “A 500-point Fourier transform using charge-coupled devices,” in *1975 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, 1975, vol. 18, pp. 144–145.

[197] Q. Shao *et al.*, “Role of dimensional crossover on spin-orbit torque efficiency in magnetic insulator thin films,” *Nat. Commun.*, vol. 9, no. 1, pp. 1–7, 2018.

[198] L. You *et al.*, “Switching of perpendicularly polarized nanomagnets with spin orbit torque without an external magnetic field by engineering a tilted anisotropy,” *Proc. Natl. Acad. Sci.*, vol. 112, no. 33, pp. 10310–10315, 2015.

[199] C. Burrowes *et al.*, “Non-adiabatic spin-torques in narrow magnetic domain walls,”

Nat. Phys., vol. 6, no. 1, pp. 17–21, 2010.

[200] T. Koyama *et al.*, “Observation of the intrinsic pinning of a magnetic domain wall in a ferromagnetic nanowire,” *Nat. Mater.*, vol. 10, no. 3, pp. 194–197, 2011.

[201] A. Thiaville, Y. Nakatani, J. Miltat, and Y. Suzuki, “Micromagnetic understanding of current-driven domain wall motion in patterned nanowires,” *EPL (Europhysics Lett.*, vol. 69, no. 6, p. 990, 2005.

[202] S. Fukami *et al.*, “Current-induced domain wall motion in perpendicularly magnetized CoFeB nanowire,” *Appl. Phys. Lett.*, vol. 98, no. 8, p. 82504, 2011.

[203] S. Parkin and S.-H. Yang, “Memory on the racetrack,” *Nat. Nanotechnol.*, vol. 10, no. 3, pp. 195–198, 2015.

[204] E. H. Hall, “On a new action of the magnet on electric currents,” *Am. J. Math.*, vol. 2, no. 3, pp. 287–292, 1879.

[205] E. H. Hall, “On the possibility of transvers currents in ferromagnets,” *Philos. Mag*, vol. 12, pp. 157–172, 1881.

[206] J. E. Hirsch, “Spin hall effect,” *Phys. Rev. Lett.*, vol. 83, no. 9, p. 1834, 1999.

[207] W. Jiang *et al.*, “Direct observation of the skyrmion Hall effect,” *Nat. Phys.*, vol. 13, no. 2, pp. 162–169, 2017.

[208] G. Yu *et al.*, “Room-temperature skyrmion shift device for memory application,” *Nano Lett.*, vol. 17, no. 1, pp. 261–268, 2017.

[209] G. Yu *et al.*, “Room-temperature creation and spin–orbit torque manipulation of skyrmions in thin films with engineered asymmetry,” *Nano Lett.*, vol. 16, no. 3, pp. 1981–1988, 2016.