Energy-Efficient Architectural Design of High-Dimensional Computing Paradigm for Edge Devices

Submitted to
Graduate School of Information Science and Technology
Osaka University

July 2024

Dehua LIANG

Publications

Journal Articles (Refereed)

[J1] Dehua Liang, Jun Shiomi, Noriyuki Miura, Masanori Hashimoto, and Hiromitsu Awano, "A Hardware Efficient Reservoir Computing System Using Cellular Automata and Ensemble Bloom Filter," *IEICE Transactions on Information and Systems*, Vol. E105-D, No. 7, pp. 1273–1282, July 2022.

[J2] Dehua Liang, Jun Shiomi, Noriyuki Miura, and Hiromitsu Awano, "StrideHD: A Binary Hyperdimensional Computing System Utilizing Window Striding for Image Classification," *IEEE Open Journal of Circuits and Systems*, just accepted (May 2024). https://doi.org/10.1109/ojcas.2024.3401028

[J3] Dehua Liang, Hiromitsu Awano, Noriyuki Miura, and Jun Shiomi, "A Robust and Energy Efficient Hyperdimensional Computing System for Voltage-scaled Circuits," *ACM Transactions on Embedded Computing Systems*, just accepted (September 2023). https://doi.org/10.1145/3620671

International Conference Papers (Refereed)

[C1] Dehua Liang, Masanori Hashimoto, and Hiromitsu Awano, "BloomCA: A Memory Efficient Reservoir Computing Hardware Implementation Using Cellular Automata and Ensemble Bloom Filter," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2021, pp. 587-590, doi: 10.23919/DATE51398.2021.9474047.

[C2] Dehua Liang, Jun Shiomi, Noriyuki Miura, and Hiromitsu Awano, "DistriHD: A Memory Efficient Distributed Binary Hyperdimensional Computing Architecture for Image Classification," in *Proceedings of the 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 43-49, Jan. 2022.

[C3] Dehua Liang, Hiromitsu Awano, Noriyuki Miura, and Jun Shiomi, "DependableHD: A Hyperdimensional Learning Framework for Edge-oriented Voltage-scaled Circuits," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 416-422, Jan. 2023.

Summary

With the emergence of the Internet of Things (IoT), devices are generating massive data streams. Running big data processing algorithms, e.g., machine learning, on edge devices poses substantial technical challenges due to limited device resources. Compared to the sophisticated machine learning method, the brain-inspired high-dimensional computing paradigm are considered as promising alternative in terms of energy efficiency and robustness, which is suitable for the resource limited scenario. As a novel computing paradigm, how to reduce the hardware cost while maintaining sufficient accuracy performance on edge devices is still an open problems for circuit designers. The goal of this thesis is thus to provide the strategies to design an energy efficient high-dimensional computing paradigm on edge devices. In this thesis, we provide the solutions for high-dimensional computing from different perspectives: arithmetic operations, memory usage, and robustness, which eventually leading to the pursuit of energy efficiency.

In Chapter 3, we discussed the bottleneck for the reservoir computing (RC) systems in several related papers, which is one of the typical high-dimensional computing paradigms. Suffered from the huge memory usage and expensive arithmetic operations, there was still gap between the RC concept to practical implementation on edge devices. Therefore, this thesis propose a novel RC architecture *EnsembleBloomCA*, which utilizes cellular automata (CA) and an ensemble Bloom filter to organize an RC system. By adopting CA as the reservoir in the RC system, it can be implemented using only binary operations and is thus energy efficient. The rich pattern dynamics created by CA can map the original input into a high-dimensional space and provide more features for the classifier. Applying the ensemble Bloom filters as the classifier, the features provided by the reservoir can be effectively memorized. As the combination of these two techniques, the novel RC architecture successfully eliminates all floating-point calculation and integer multiplication. Our experiment result demonstrated that $43\times$ and $8.5\times$ reduction is achieved in terms of memory usage and power consumption, while the accuracy performance is maintained.

Although the extreme energy efficiency is achieved in the first prototype with novel RC architecture, the circuits designer's target changes for variety application scenarios. The high-dimensional computing paradigm are also expected to achieve not only

sufficient but also competitive accuracy performance, which has posed designers problems to balance the trade-off between different key indicators. In Chapter 4, we found the similar issues should be tackled in hyper-dimensional computing (HDC), which are also considered as high-dimensional computing paradigm. To alleviate the huge memory cost during encoding procedure in HDC (i.e., over 95% of the memory capacity is consumed), we propose a novel HDC architecture *StrideHD* that utilizes the window striding in image classification. It encodes data points to distributed binary hypervectors and eliminates the expensive Channel item Memory (CiM) and item Memory (iM) in the encoder, which significantly reduces the required hardware cost for inference. For the improvement of accuracy on edge devices during inference, we provide the iterative learning mode for the proposed HDC architecture. It also enables HDC systems to be trained and tested using binary hypervectors and achieves very competitive accuracy performance. The experiment result shows that the proposed HDC model achieves extreme memory efficiency (27.6×) with acceptable accuracy performance under the single-pass mode, while its iterative mode provides competitive performance (11.33% classification accuracy improvement) and keeping the memory efficiency (8.7*times* improvement) for inference phase. The iterative retraining can be accomplished within fewer iterations compared to the baseline HDC works.

Besides the trade-off between accuracy performance and hardware resource in high-dimensional computing, the robustness issue are also arousing the attention of circuit designers. As we commonly known, scaling down the supply voltage is a promising approach to reduce the energy consumption of the circuits, while the aggressive voltage scaling will pose designers several severe problems such as the performance variation and functional failures. It is a potential solution to tolerate the voltage-scaling induced functional failures by the superior robustness of the brain-inspired high-dimensional computing paradigms. Therefore, we introduces the concept of margin enhancement for model retraining and utilizes noise injection to improve the robustness in the proposed HDC framework *DependableHD*, which is capable of application in most state-of-the-art HDC algorithms. After analyzing the error patterns in voltage-scaled circuits, we come up with a strategy to take fully advantage of the equivalent structure transformation in HDC architecture. We additionally propose the dimension-swapping technique, which aims at handling the stuck-at errors induced by aggressive voltage scaling in the memory cells. The experiment shows that under 8% memory stuck-at error, the proposed method exhibits a 2.42% accuracy loss on average, which achieves a 14.1× robustness improvement compared to the baseline HDC solution. The work also supports the systems to reduce the supply voltage from 430mV to 340mV for both item Memory and Associative Memory, which provides a 41.8% energy consumption reduction while maintaining competitive accuracy performance.

Acknowledgments

First of all, I would like to express my deepest gratitude to Associate Professor Jun Shiomi for his invaluable guidance, support, and encouragement throughout my PhD journey at Osaka University. His profound expertise, insightful suggestions, and advanced perspective have led me to these achievements. His mentorship has been a cornerstone of my research career, and I am sincerely appreciative of all the time and effort he has invested in my development.

I would like to express my sincere appreciation to Associate Professor Hiromitsu Awano at Kyoto University for his precious suggestions and enormous help throughout my master's and doctoral research.

I am deeply grateful to Professor Noriyuki Miura at Osaka University for providing me with a precious opportunity and an excellent environment to study as a doctoral student in his laboratory.

I would also like to thank Professor Yasushi Sakurai at Osaka University for his detailed reviews and insightful suggestions.

I extend my appreciation to Professor Masanori Hashimoto at Kyoto University for giving me the chance to study VLSI design in his laboratory during my master's degree.

My sincere appreciation goes to Associate Professor Yoshihiro Midoh at Osaka University, Assistant Professor Ryo Shirai at Kyoto University, and Associate Professor Jaehoon Yu at the Tokyo Institute of Technology for their invaluable suggestions and enormous help.

I am grateful to Dr. Jun Chen, Dr. Tai-Yu Cheng, Miss Yangchao Zhang, Mr. Lukas Nakamura from Osaka University and Dr. Ángel López from Tokyo Institute of Technology for their technical discussions and suggestions. I would also like to thank my colleagues, past and present, from the Intelligent Integrated Systems Laboratory at Osaka University for their daily discussions and support. Special thanks to our laboratory secretary, Mrs. Makiko Arai, for her various support. I appreciate the financial support from the Osaka University fellowship. Additionally, I would like to thank all of my friends for their financial and emotional support.

I would like to extend my heartfelt gratitude to my parents. They have always supported and encouraged me with their best wishes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation aims to provide an energy-efficient architectural design of the high-dimensional computing paradigms for edge devices. This chapter provides the background of the research and the objectives of this dissertation. Section 1.1 introduces the background of AI applications for edge devices. Section 1.2 shows the motivation of high-dimensional computing paradigm. Section 1.3 describes the challenges for this research and points out issues preventing us from energy-efficient hardware implementation. Section 1.4 claims the goal of our research and the contribution of this thesis. Finally, the overall organization of this dissertation is presented in Section 1.5.

1.1 Background

Artificial intelligence (AI) has become a ubiquitous technology indispensable for personal daily life. Statistical data reveals that the investment in AI has incredibly grown in recent years, which could even burst in the coming decade [1]. Fig 1.1 shows that the AI market has grown to 208 billion USD by 2023, while the data forecast that the value would reach 1.85 trillion USD by 2030. Currently, the AI market covers a vast number of industries, such as IoT, supply chains, and product making, while more are fields that will in some aspect adopt artificial intelligence within their business structures. Chatbots, image/video generating AI, IoT, and mobile applications are all among the major trends improving AI in the near future.

Rather than serving as a replacement for human intelligence, AI should be considered a supporting tool for our society in the information era. The machine learning algorithms in AI applications can recognize objects and speech and have mastered games like chess, even surpassing human performance, *e.g.*, DeepMind's AlphaGo Zero [2]. More and more AI-related companies are poised to play a crucial role in shaping the future economy and society. Taking the OpenAI as an example, it stands among the

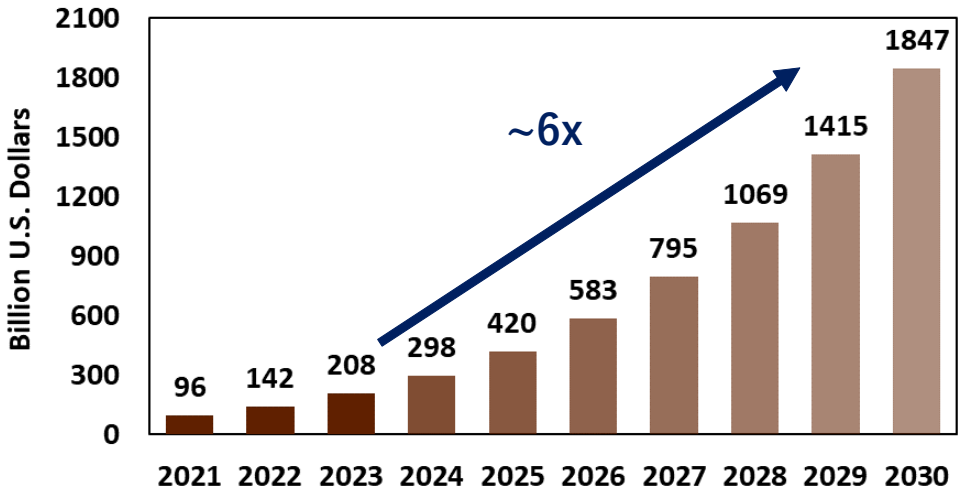

Figure 1.1: The worldwide market size of Artificial Intelligence (AI) in 2021 with a forecast until 2030 (in billion U.S. dollars).

most well-funded machine learning related startups globally, having secured more than 12 billion USD in investments and recorded more than 100 million monthly active users as of 2023.

Deep neural network (DNN) has thrived and formed the foundation for many advanced machine learning algorithms in the AI applications due to its unmatchable performance. It contributes to unprecedented success in different AI tasks thanks to its outstanding accuracy and generality in several fields, *e.g.*, image processing, computer vision, and natural language processing. In many of these domains, DNNs are now able to exceed human accuracy performance, which mainly comes from its ability to extract high-level features from raw sensory data after using statistical learning over a large amount of data to obtain an effective representation of an input space. This is different from earlier approaches that use hand-crafted features or rules designed by experts [3]. However, as DNNs grow in complexity, their associated energy consumption becomes a challenging problem when it comes to the implementation in edge devices like the Internet of Things (IoT).

Villalobos *et al.* study the tendency in model size of the popular and notable machine learning systems over time in Ref. [4]. A great increase shows in the size of machine learning models, measured in the number of free parameters that have to be fit to the data. Model size has become especially important as researchers have improved the understanding of scaling laws for language models, which govern how increases in model size and training data produce better performance. However, the rapid growth of model size and computing resource requirement also leads to the bottleneck of machine learning on edge devices. In general, DNNs have millions of parameters. For exam-

ple, a popular DNN model, AlexNet [5] has 60 million parameters and requires 249MB of inference memory and performs 1.5 billion high-precision operations to classify one image. More recently, it took seven months to train a DNN with 175 billion parameters, developed by OpenAI for natural language processing (NLP) [6]. The memory capacity of most edge devices is hard to store all data for large-size DNNs with hundreds of layers and millions of weights. Even applying the hardware-friendly implementation techniques to get the Binarized Neural Networks (BNNs) [7] or XNOR-Networks [8], still require considerably intensive computation costs due to the floating-point calculation and backpropagation algorithm during the training. The emergence of the IoT has led to a copious amount of small connected edge-oriented devices and systems [9], while most of these small edge-oriented systems do not have sufficient computing power to accomplish the sophisticated machine learning algorithms such as DNNs individually. The energy constraint of edge devices hinders them from the real-time training of neural network models [10].

In order to optimize the power dissipation of digital systems on edge devices, low-power methodology should be applied throughout the design process, which usually includes the system-level design, architecture-level design, and device-level design [11].

Regarding to the device-level design, one of the most practical ways of reducing the energy consumption for edge devices is to shrink the feature size of transistors, *i.e.*, the minimum length of the MOS transistor channel between the drain and the source. Moore's law has been driving the semiconductor industry for over 50 years. With continuously scaling of technology node, the transistor count per die kept doubling in bi-annual pace [12–15]. Meanwhile, though clock frequency for single thread reached operation has reached a plateau around the year of 2006, it still keeps 15% to 20% increment per technology node generation [13, 16]. Following the ITRS2.0 (International Technology Roadmap for Semiconductors) and IRDS (International Roadmap for Devices and Systems) prediction [17, 18], to the 2030s, even for the low-power mobile devices, computing capability can increase $10\times$ within a decade. However, such a technology scaling trend pushes the future chip design to the power wall [14, 19, 20], because the increasing frequency and transistor count will eventually hit the physical limitations such as thermal dissipation limit and battery capacity limit [21]. Fig. 1.2 shows the trend data of the commercial microprocessor in terms of number of transistors and maximum working clock frequency, which are collected by M. Horowitz *et al.* in Ref. [12, 13]. Therefore, more and more researchers turn their attention to the exploration of system-level design and architectural design for driving the computer performance in the Post-Moore's Era.

In terms of the architecture-level optimization, many designers needs to balance the trade-off between efficiency and generality base on the application scenario. Compared to the general-purpose Central Processing Units (CPUs), many AI applications

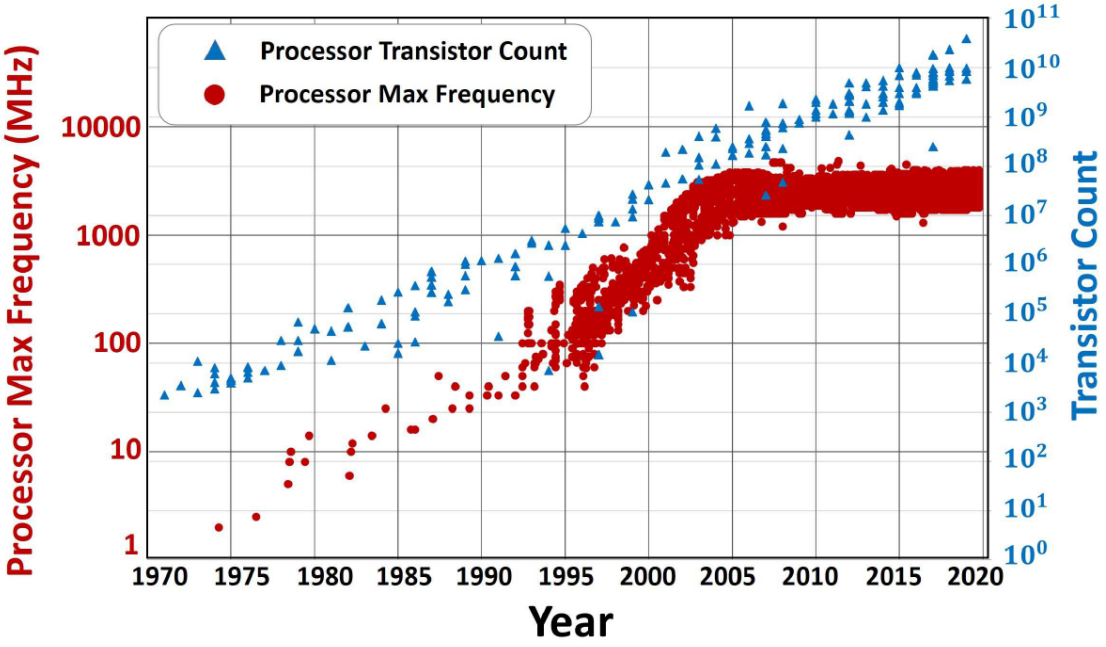

Figure 1.2: Trend Data of the Transistors Numbers and Maximum Clock Frequency in Microprocessor.

look to the Graphics Processing Units (GPUs) and Tensor Processing Unit (TPUs) as accelerator architectures. GPUs, initially designed for graphics rendering, have evolved into versatile processors capable of handling AI tasks due to their parallel processing strengths. For the TPUs developed by Google, they are specialized for AI computations, and offer optimized performance for many popular machine learning algorithms. When it comes to implementation on edge devices, the Application-Specific Integrated Circuits (ASICs) are customized for a particular use and leads to the extreme energy efficiency. Rather than intended for general-purpose use, ASICs are optimized to deliver the best performance for a specific application, making them incredibly efficient but inflexible compared to CPUs, GPUs, and TPUs. The co-design between system-level algorithms and architecture-level hardware implementation provides potential solution for the strict resource constrains on edge devices.

For the system-level design, the power reduction can be achieved through algorithm selection [11]. Cloud computing has emerged as one of the solutions to meet the hardware requirements of complex models. It has become increasingly common to deploy the ML algorithms like DNNs using cloud platforms, which are high-performance computing platforms with tremendous speed and memory. Training can be performed within a reasonable time on cloud machine learning platforms, where the servers provide large computational resources, large data storage, high-speed computation, low latency, and high availability. Cloud computing is also used to deploy DNNs for on-

line applications [22]. However, cloud computing also causes high latency and requires high transmission bandwidth and privacy security issues [23]. For example, in health care monitoring, we often require learning algorithms to have real-time control of the patient's daily behavior, speech, and bio-medical sensors. Sending all data points to the cloud, cannot guarantee scalability and real-time response, which is often undesirable due to privacy and security concerns [24]. On the other hand, edge computing aims at processing the information/data from edge devices with extreme limited resource constrains. For the specific scenario, the processing task with simple workload is requested on the edge devices, which makes it possible to run the light-weight model and finish the task on local devices. Hence, for running the machine learning algorithms on edge devices, we recognize the following technical challenges:

- **Sufficient performance for edge applications:** Although trade-offs may exist in machine learning algorithms between factors such as accuracy, speed, and resource consumption, maintaining high accuracy remains a primary objective when deploying AI solutions in real-life scenarios.
- **Limited hardware resources:** The embedded devices in IoT systems often do not have sufficient computing resources and memory capacity for the processing of sophisticated learning and big data applications. Running existing machine learning using traditional cores results in high energy consumption.
- **High robustness:** The technological and fabrication issues in highly scaled technology nodes add a significant amount of noise to the VLSI circuits. Most existing algorithms do not have the enough robustness to work with noisy devices while providing accurate results.

To address such challenges for edge devices, the demand for a more energy-efficient architectural design with the novel computing paradigms is crucially rising.

1.2 Motivation: High-Dimensional Computing

Classical computation can be interpreted as the computers embodied in the Turing/von Neumann paradigm. Nowadays it has been incredibly successful for information processing. However, the increased demand due to highly complex computational tasks has motivated the search for advanced unconventional computation. To achieve real-time performance with high energy efficiency, we need to rethink not only how we accelerate machine learning algorithms in hardware, but also we need to redesign the algorithms themselves using strategies that more closely model the ultimate efficient learning machine: *the human brain*. The nervous system in human brains carries out computation through elegant and sophisticated spatio-temporal dynamics [25]. The human brains are capable to process signals with excellent precision and power efficiency (the brain drains 20% of the body's energy [26, 27], which amounts to an overall power-consumption es-

timated at around 20 W). Such energy requirements are impressive compared to digital machines, which easily exceed 100 W during inference and several kilowatts during the training of traditional DNNs. Motivated by the observation that the human brain operates on high dimensional representations of data originated from the large size of brain circuits [28], computational neuroscientists begins the exploration of modeling memory of human brain. In the high-dimensional computing (e.g., reservoir computing, hyperdimensional computing), human memory are modeled by mapping input data points to the representation in high-dimensional space, while the important functionalities of human brain are refered by the well-defined vector operations.

Reservoir computing (RC), first applied to temporal signal processing, is a recurrent neural network in which neurons are randomly connected. Once initialized, the connection strengths remain unchanged. These fixed and nonlinear hidden layers are generally referred to as the *reservoir* part, which can map the original input data into a higher-dimensional feature space. On the other hand, the output layer is considered as a *classifier*. The model's rich dynamics, linear separability, and memory capacity then enable a simple linear readout to generate adequate responses for various applications [29]. Compared with conventional DNNs, RC systems use fixed weights in the input and hidden layers and only modify the weights of the output layer in the training process [30]. Because most of the weights can be fixed during training and inference, they can be implemented using hardwired logic and thus do not require memory circuits. Hence, RC is considered to be a promising alternative to replace DNNs, the model size of which continues to increase exponentially [31].

Hyperdimensional computing (HDC) is a strategy developed by computational neuroscientists as a model the human short-term memory. HDC is motivated by the understanding that the human brain operates on high dimensional representations of data originated from the large size of brain circuits. It models the human memory using points of a high-dimensional space, called *hypervectors*. The hyperspace typically refers to tens of thousand dimensions. HD mimics several important functionalities of the human memory model with vector operations which are computationally tractable and mathematically rigorou. HDC models are computationally efficient with high parallelism and amenable to hardware level optimization. It offers a complete computational paradigm that can be applied to cognitive as well as learning problems. Meanwhile, it provides strong robustness to noise and naturally enable secure and lightweight learning, which makes it a promising solution for today's edge devices with limited storage, battery, and resources, as well as future computing systems in deep nano-scaled technology which devices will have high noise and variability.

One one hand, both of RC and HDC have been widely explored by many researchers. These two kind of systems are connected by several similar core principles in the current studies: (i) Random projections of input values onto a reservoir (which in essence

is a high-dimensional vector) matches random HDC representations stored in a superposition. (ii) The update of the reservoir by a random recurrent connection matrix is similar to HDC adding/shifting operations. (iii) The non-linearity of the reservoir can be approximated with the thresholded addition of integers in HDC. Therefore, both of RC and HDC can be considered as high-dimensional computing paradigms.

On the other hand, the focus of these two types of computing systems might be different for the current studies. For the RC system using reservoirs based on physical phenomena has recently attracted increasing interest in many research areas. Various physical systems, substrates, and devices have been proposed for realizing RC. A motivation for physical implementation of reservoirs is to realize fast information processing devices with low learning cost. Regarding the HDC system with massive parallelism and simple arithmetic, different kinds of training strategies and architectures have been proposed for the scope of energy-efficient and ultra-low-latency computing, especially with the rise of emerging hardware.

As frameworks for neural symbolic representation, computation, and analogical reasoning, there are two main advantages for the high-dimensional computing paradigms compared to the conventional DNN architectures: (i) Only some of the parameters are required to be trained, while the rest can be fixed. (ii) The complex data structures and analogical reasoning are implemented by simple arithmetical operations, *e.g.*, binding, addition/bundling, and permutation, and a well defined similarity metric. Owing to these unique features, the RC systems and HDC systems can be implemented with limited hardware resources.

Besides the novel computing paradigm, the voltage-scaling is also one of the classic low power design methodologies for energy efficiency improvement in device-level. In typical circuits, the dominant source of energy consumption is a dynamic energy consumption which is consumed when Complementary MOS (CMOS) circuits charge or discharge load capacitors. Since the dynamic energy consumption is quadratically proportional to the supply voltage, the low voltage operation is a good choice to reduce the power and energy dissipation at the cost of degrading the computational power [32]. Although the it also brings challenges to fully guaranteeing stable operation in the circuits, the natural robustness of HDC provides the room for the systems to tolerate the hardware error issues induced by voltage-scaling.

Therefore, in order to achieve practical machine learning on edge devices, we propose utilizing high-dimensional computing paradigm with energy-efficient architectural design, which includes three techniques from different design level: (i) reservoir computing for light-weight implementation, (ii) hyper-dimensional computing for energy reduction, and (iii) voltage-scaling for low power design. The literature review and preliminaries of these techniques are discussed in the Chapter 2.

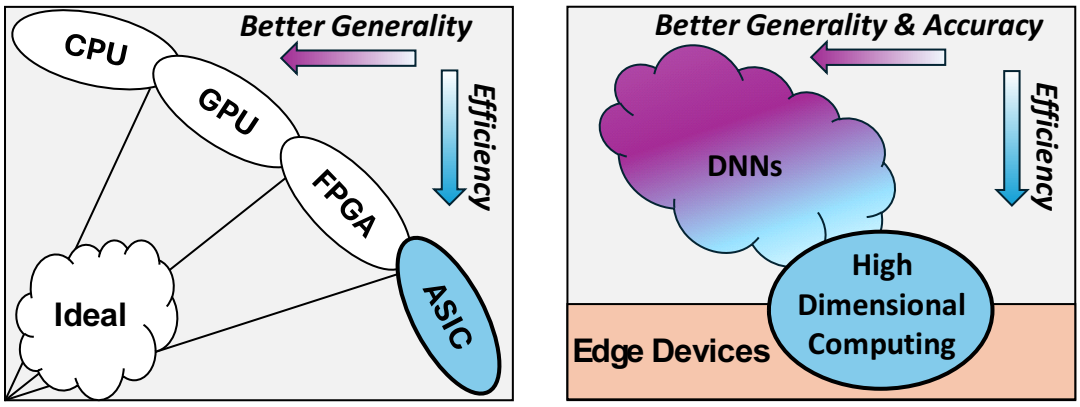

Figure 1.3: (a) Generality and Energy Efficiency of Computing Architecture. (b) Generality/Accuracy and Efficiency of Computing Paradigm.

1.3 Challenges toward High-Dimensional Computing

Fig. 1.3 shows the research target area of this thesis. Fig. 1.3(a) shows the generality and energy efficiency of different computing architectures. CPUs are the most general computing architecture. However, their computing efficiency is not sufficient for some edge computing scenarios. Other popular architectures are GPUs and FPGAs at the cost of degrading the generality, while ASICs are the most efficient device for specific dedicated purposes. From the viewpoint of edge computing, ASICs are a suitable candidate for lightweight specific purposes in edge computing. Fig. 1.3(b) shows the generality/accuracy and energy efficiency of different computing paradigms. Here the generality intents to represent the ability of handling multi-task. Many other popular machine learning algorithms (*e.g.*, multi-task learning) has been emerged as a powerful paradigm in deep learning to obtain language and visual representations from large-scale data [33]. By leveraging supervised data from related tasks, these approaches reduce the expensive cost of curating the massive per-task training data sets needed by deep learning methods and provide a shared representation which is also more efficient for learning over multiple tasks. When targeting the scenarios with multiple tasks, the current researches about high-dimensional computing paradigms are still lack of convincing performance in terms of generality. Most high-dimensional computing paradigms are focusing on the energy efficiency and robustness improvement. The ability of handling single task with wide coverage is sufficient for the scenarios on edge devices. The DNNs are very powerful but energy consuming, which may not be suitable for the edge computing scenario. Therefore, this thesis targets high-dimensional computing with ASICs as a potential solution for the edge computing scenario. This section describes challenges toward high-dimensional computing.

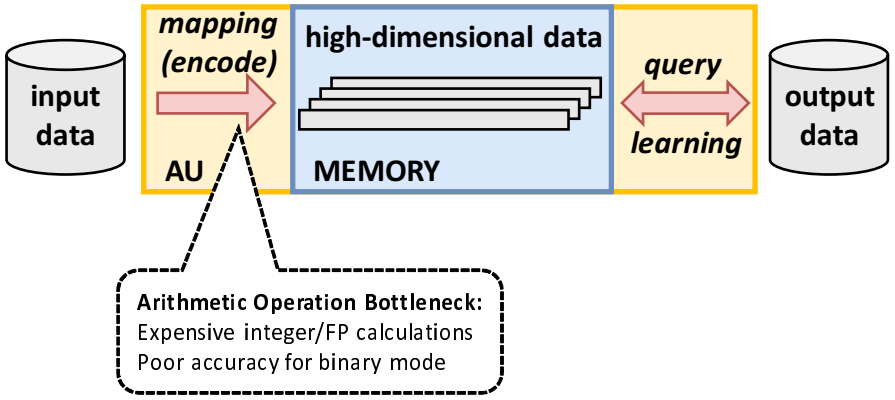

Figure 1.4: Arithmetic Operations Challenges on High-Dimensional Computing.

1.3.1 Elimination of Expensive Arithmetic Operations

Although the high-dimensional computing is a promising alternative for drastically reducing the computational burden compared to the traditional machine learning methods, the trade-off between accuracy performance and arithmetic precision still exists. From the circuit designer's point of view, it is important to reduce the hardware cost while maintaining accuracy performance. The arithmetic operations challenges on high-dimensional computing paradigms are diagrammed in Fig. 1.4. For high-dimensional computing, the hardware cost of arithmetic operations is affected by the precision of elements in the model. Although applying the precised FP calculation on high-dimensional computing can improve the accuracy performance, it also brings heavy computational burden to the edge devices.

For example, Ref. [34] select the most accurate cellular automata rules to represent the reservoir structure, which can easily be reproduced using a set of XOR gates and shift registers. They achieve a high-performance alternative for RC hardware implementation in terms of circuit area, power, and system accuracy, which can be considered as a low-cost method to implement fast pattern recognition digital circuits. However, this model exploits the softmax function as its classifier, which still requires expensive floating-point (FP) calculations; hence, it is not suitable for implementation on resource-constrained edge devices. For [35], authors utilize a random forest algorithm as the classifier, which avoids the costly FP calculation during the inference but also brings higher consumption for the training.

Similarly, in some HDC works, it is common to utilize hypervector with high-precision elements for accuracy improvement. In Ref. [36–41], authors need to encode data points to hypervectors with non-binary elements, *i.e.*, storing integer or FP value for each element. This leads to the high memory requirement and expensive computational cost. To reduce the inference cost, Rahimi *et al.* [42] and Imani *et al.* [24] propose bina-

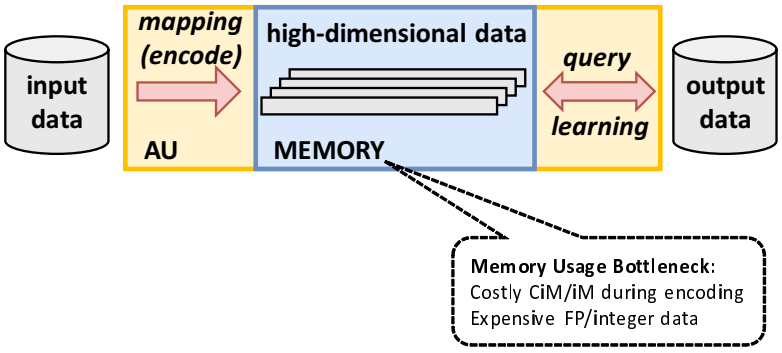

Figure 1.5: Memory Usage Challenges on High-Dimensional Computing.

rizing the elements of class hypervectors after the training. Although these approaches can simplify the inference similarity matrix to Hamming distance and lead to a faster computation speed, it comes with the cost of significantly degraded HD classification accuracy on practical image recognition applications. For instance, Imani *et al.* [40] proposed a HDC model for face image classification task. If the model is binarized, the accuracy performance sharply decreases to 38.9%, which is far lower than the non-binarized mode. Since the frequent use of floating-point and integer calculations makes the hardware implementation challenging, the elimination of expensive arithmetic operations while maintaining accuracy performance becomes a must.

1.3.2 Memory Efficient High-Dimensional Computing

The high-dimensional computing paradigms need to ensure that the model has the separation capacity so that the no-linear mapping operations allow to differentiate different input signals. The performance of the computing paradigms usually related to how well the model separates the input signals, i.e. whether sufficiently different input signals become linearly separable in the high-dimensional space. The stronger a model's separation capacity, the larger the set of functions that it can approximate, which generally increases with the size of the no-linearity of networks, and also grows with larger memory capacity. The memory efficiency challenges on high-dimensional computing paradigms are diagrammed in Fig. 1.5. For high-dimensional computing, the input data points are mapped into the high-dimensional space in a fixed and no-linear way, which usually takes huge memory usage. Also, to storage the universal information in high-dimensional space, the hardware cost sharply increase when utilizing high-precision elements.

For example, Ref. [43, 44] adopts the bloom filters as the classifier in the RC system, which is a space-efficient probabilistic data structure aimed at approximate member queries. Such structure is capable of eliminating the costly FP calculations and characterized by its simple implementation in both software and hardware. However, the large

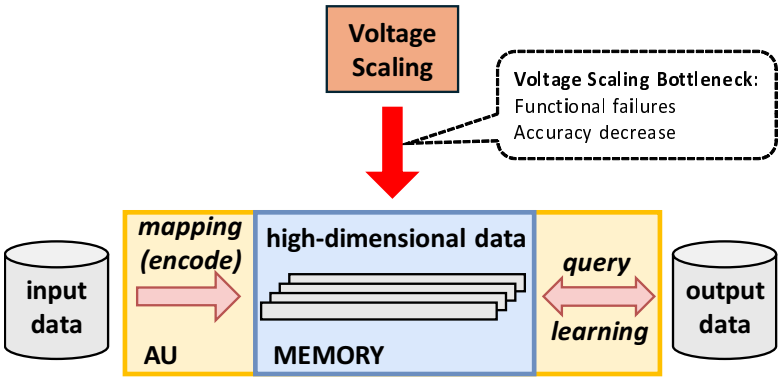

Figure 1.6: Voltage Scaling Challenges on High-Dimensional Computing.

amount of memory required remains as the bottleneck for its application, which makes it impractical when used for a portable device or hardware-resource-constrained system. Also, in many of the popular HDC models [36–42, 45], they need to represent the index and value of the input features via hypervectors, which requires a large size of memory blocks. Such a process takes huge occupation of the total inference memory cost, e.g., 96.15% for a letters recognition task. Therefore, the demand of memory efficiency improvement is crucially rising for the high-dimensional computing paradigms.

1.3.3 Model Robustness in the Low Voltage Operation

The voltage scaling challenges on high-dimensional computing paradigms are diagrammed in Fig. 1.6. The aggressively scaled voltage operation poses the further requirement of model robustness. Energy consumption in circuits largely results from the charging and discharging of internal node capacitances and can be reduced quadratically by lowering supply voltage (V_{DD}). Ref. [46] achieved 4.7× energy consumption reduction by scaling down the voltage to the near-threshold voltage region compared with the nominal voltage operation. The near-threshold voltage region is a voltage region where the supply voltage is downscaled to near the threshold voltage of the transistor. If we further downscale the supply voltage, the minimum energy voltage can be found in the sub-threshold region [47], where the supply voltage is reduced below the threshold voltage of the transistor [48]. However, low-voltage designs may suffer from functional failures due to various causes, e.g., soft errors, aging, and processing variation. As a result, such functional failures might eventually result in the degradation of performance on edge devices.

A key attribute of high-dimensional computing is its robustness to the imperfections associated with the computational substrates on which it is implemented. It is therefore particularly amenable to emerging non-von Neumann approaches such as in-memory computing, where the physical attributes of nanoscale memristive devices are exploited

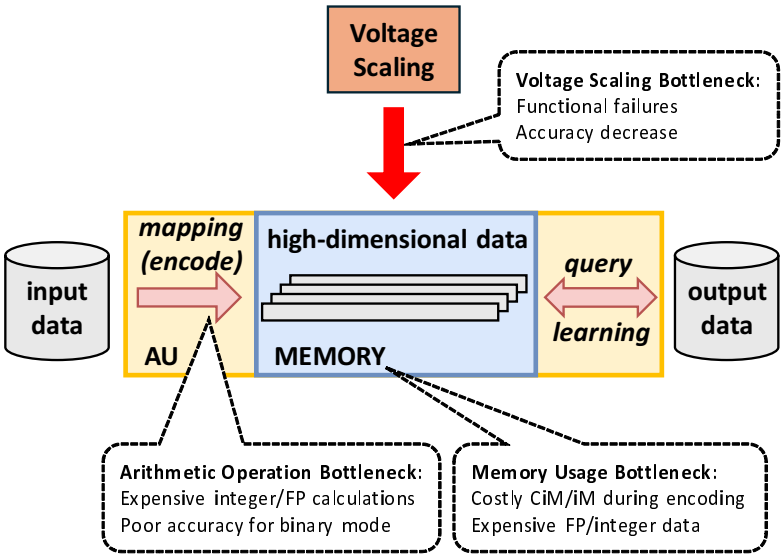

Figure 1.7: Overview of the Challenges on High-Dimensional Computing.

to perform computation [49]. Many works have indicated and explored the robustness of HDC learning framework. Ref. [42, 50] proposed a robust HDC that can tolerate approximately 5×10^{-7} probability of failure of memory cells while maintaining accuracy. Compared to the conventional DNNs architecture, the robustness of high-dimensional computing not only brings the advantage toward the future computing systems in deep nano-scaled technology which devices will have high noise and variability but also provides the potential for voltage-scaling on edge devices [51–54]. However, it is still far away from the prior solution to serious memory failure issues induced by scaling down the voltage to the sub-threshold region. For example, if the supply voltage of 65nm SRAM cells are scaled down from normal value (1.2V) to 500mV, the typical failure probability sharply increases from $\sim 10^{-7}$ to 4% due to the process variations [55, 56]. Therefore, a further development of model robustness is necessary for applying aggressive voltage-scaling technique on edge devices.

1.4 Research Goal and Thesis Contribution

1.4.1 Research Target

The goal of this thesis is to improve the energy efficiency for the architectural design of high-dimensional computing. Although the high-dimensional computing paradigm is a promising approach for the implementation on edge devices, its application still suffers from the expensive arithmetic operations, huge memory cost, and insufficient robustness against the aggressive voltage-scaling. The overall challenges on high-dimensional

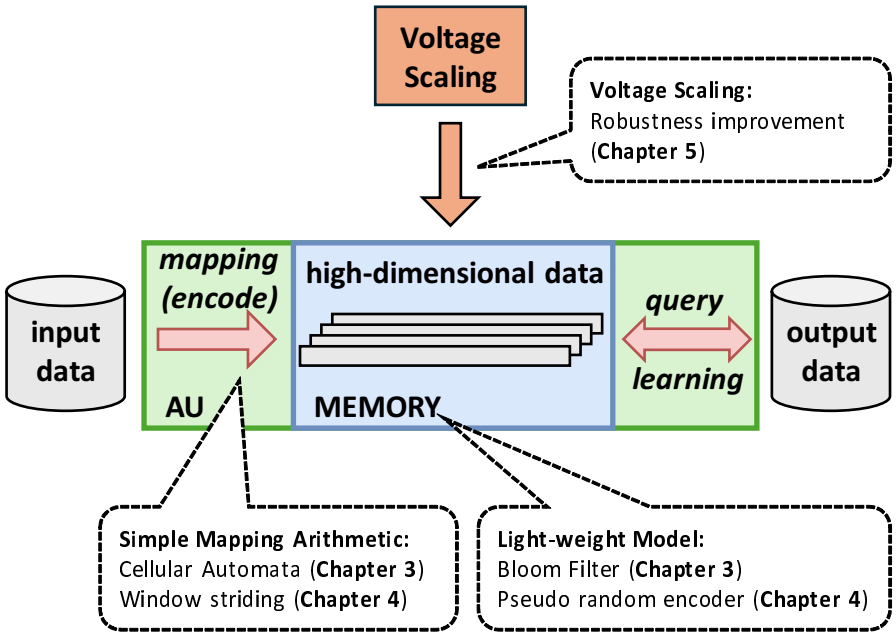

Figure 1.8: Solutions toward the Challenges on High-Dimensional Computing.

computing paradigms are diagrammed in Fig. 1.7. The following requirements should be met for achieving energy-efficient high-dimensional computing architecture on edge devices:

1. Capability of hardware-friendly arithmetic operations,
2. Architecture with high memory efficiency, and
3. Model robustness in against to functional failures in voltage over-scaling.

From the circuit designer's point of view, the first two items are considered as universal and key indicators. The high-dimensional computing paradigm needs the capability to choose the appropriate arithmetic operations for different scenarios. Although the application of voltage-scaling has been widely studied, how to utilize model robustness to tolerate the functional failures induced by aggressive voltage-scaling is still the open problem.

1.4.2 Thesis Contribution

To pursue the research target, this thesis presents an RC system with extreme hardware efficiency, an HDC system with competitive accuracy performance and memory efficiency, as well as the strategies overcoming the functional failures when applying voltage over-scaling. The solutions toward the challenges on high-dimensional computing paradigms are diagrammed in Fig. 1.8. The contribution of this thesis is summarized below.

- ***EnsembleBloomCA*: An RC System for Light-weight Implementation**

Reservoir computing (RC) is an attractive alternative to machine learning models owing to its computationally inexpensive training process and simplicity. In this thesis, we propose *EnsembleBloomCA*, which utilizes cellular automata (CA) and an ensemble Bloom filter to organize an RC system. In contrast to most existing RC systems, *EnsembleBloomCA* eliminates all floating-point calculation and integer multiplication. *EnsembleBloomCA* adopts CA as the reservoir in the RC system because it can be implemented using only binary operations and is thus energy efficient. The rich pattern dynamics created by CA can map the original input into a high-dimensional space and provide more features for the classifier. Utilizing an ensemble Bloom filter as the classifier, the features provided by the reservoir can be effectively memorized. Our experiment revealed that applying the ensemble mechanism to the Bloom filter resulted in a significant reduction in memory cost during the inference phase. In comparison with *Bloom WiSARD*, one of the state-of-the-art reference work, the *EnsembleBloomCA* model achieves a 43× reduction in memory cost while maintaining the same accuracy. Our hardware implementation also demonstrated that *EnsembleBloomCA* achieved over 23× and 8.5× reductions in area and power, respectively.
Such architecture is suitable for scenarios targeting medium-level tasks with extremely strict resource constraints. Because it can eliminate the costly arithmetic operations, supporting the hardware-friendly bit-wise operations for extreme energy efficiency. With around 18 KB memory usage, 91.9% classification accuracy is achieved for the handwritten number recognition task, which can be considered as a practical solution for edge devices with extremely limited hardware resources.

- ***StrideHD*: A Memory Efficient HDC System**
  Hyper-Dimensional (HD) computing is a brain-inspired learning approach for efficient and fast learning on today's embedded devices. Although HDC achieved reasonable performances in several practical tasks, it comes with huge memory requirements since the data point should be stored in a very long vector having thousands of bits. To alleviate this problem, we propose a novel HDC architecture, called *StrideHD*. By utilizing the window striding in image classification, *StrideHD* enables HDC system to be trained and tested using binary hypervectors and achieves high accuracy with fast training speed and significantly low hardware resources. *StrideHD* encodes data points to distributed binary hypervectors and eliminates the expensive Channel item Memory (CiM) and item Memory (iM) in the encoder, which significantly reduces the required hardware cost for inference. Our evaluation also shows that compared with two popular HD algorithms, the single-pass *StrideHD* model achieves a 27.6× and 8.2× reduction in inference memory cost without hurting the classification accuracy, while the iterative mode further provides 8.7× memory efficiency. Under the same inference memory cost,

our single-pass mode *StrideHD* averagely achieves 13.56% accuracy improvement in comparison with the single-pass baseline HD, which is a similar performance even in comparison with the costly iterative baseline HD models. As an extension, the iterative retraining mode of *StrideHD* averagely provides 11.33% accuracy improvement to its single-pass mode, which can be accomplished in fewer iterations in comparison with the baseline HD algorithms. The hardware implementation also demonstrates that *StrideHD* achieves over 9.9× and 28.8× reduction compared with baseline in area and power, respectively.
This architecture is suitable for scenarios with limited resources and facing complex datasets. It supports elements in fp16, int16, and binary format. The wide range of element precision provides sufficient room for the customers to select the appropriate strategy. The single pass mode training can achieve acceptable accuracy performance, while the iterative training mode can further provide 13% accuracy improvement. Even for the recognition tasks in complicated levels, a competitive accuracy is achieved while maintaining 8.7× memory efficiency.

- ***DependableHDv2*: A Robust HDC System for Voltage-scaled Circuits**
  Voltage scaling is one of the most promising approaches for energy efficiency improvement but also brings challenges to fully guaranteeing stable operation in modern VLSI.
  To tackle such issues, we firstly developed a *DependableHD* framework, which aims to improve the robustness of HDC in against to the functional failures induced by voltage-scaling. Targeting the stuck-at errors of SRAM cells in low voltage region, we extend the *DependableHD* to the second version *DependableHDv2* for further improvement in HDC robustness. *DependableHDv2* introduces the concept of margin enhancement for model retraining and utilizes noise injection to improve the robustness, which is capable of application in most state-of-the-art HDC algorithms. We additionally propose the dimension-swapping technique, which aims at handling the stuck-at errors induced by aggressive voltage scaling in the memory cells. Our experiment shows that under 8% memory stuck-at error, *DependableHDv2* exhibits a 2.42% accuracy loss on average, which achieves a 14.1× robustness improvement compared to the baseline HDC solution. The hardware evaluation shows that *DependableHDv2* supports the systems to reduce the supply voltage from 430mV to 340mV for both item Memory and Associative Memory, which provides a 41.8% energy consumption reduction while maintaining competitive accuracy performance.
  It is suitable for scenarios with an extremely unreliable and noisy situation. Such architecture and design strategy can tolerate more than a 10% memory error for the medium-level task. Exploiting this advantage, aggressive voltage downscaling can be applied to the systems. Note that both of margin enhancement and

random noise injection techniques in this model focus on the retraining phase, which means it can be easily applied to other HDC architectures simultaneously for robustness improvement.

1.5 Thesis Organization

This thesis is organized in the following way. In Chapter 2, we present the literature survey and preliminaries relating the energy-efficient computing paradigms and the low power design methodology, *i.e.*, reservoir computing (RC), hyper-dimensional computing, and voltage-scaling. In Chapter 3, this thesis presents an RC system with extreme hardware efficiency, which utilize cellular automata and ensemble Bloom Filter. Chapter 4 presents a binary HDC system ttilizing window striding for image classification. Chapter 5 presents the strategies overcoming the functional failures when applying voltage over-scaling in HDC system. Chapter 6 concludes this thesis.

Chapter 2

Literature Review and Preliminaries on High-Dimensional Computing

This chapter provides the literature review and preliminaries for this dissertation. Section 2.1 and 2.2 introduces the reservoir computing (RC) system and hyper-dimensional computing system, while both of these two computing paradigms aim to map the input data into high-dimensional space and can be considered as the energy-efficient alternatives. Section 2.3 shows the basic of voltage-scaling, which is a classic and promising low power design methodology. Finally, a summary of these techniques is presented in Section 2.4.

2.1 Reservoir Computing

2.1.1 Basics of Reservoir Computing

Reservoir Computing (RC) can be considered as a potential candidate of the energy-efficient computing paradigms for edge devices. Fig. 2.1 shows a structural comparison between conventional DNN models and RC systems. The most critical advantage of RC is that only some of the parameters are trained, while the rest can be fixed. Owing to this unique feature, RC can be implemented with limited hardware resources, that is, fixed weights can be realized using hardwired logic. RC circumvents the difficulty of learning a number of RNN parameters, which has been successfully applied in numerous fields such as robot control [57] and image/video processing [58]. More recently, echo state networks [59] and liquid state machines [60] have been proposed for the use in different research domains. These technologies are collectively referred to as RC because both have a component called a *reservoir* [61]. The fixed and nonlinear hidden layers are generally referred to as the *reservoir* part, which can map the original input data into a higher-dimensional feature space. Meanwhile, the output layer is considered as a

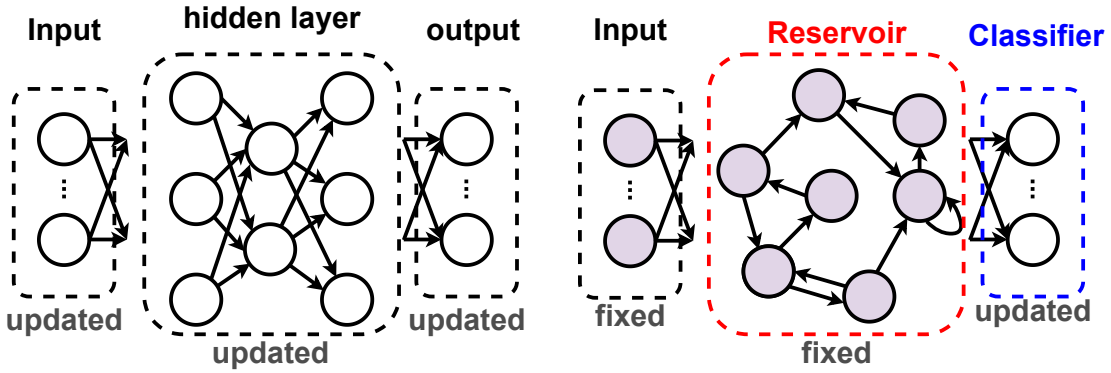

Figure 2.1: Comparison of conventional DNN and Reservoir Computing (RC) system.

classifier.

Compared with conventional neural networks, RC systems use fixed weights in the input and hidden layers and only modify the weights of the output layer in the training process [30]. Because most of the weights can be fixed during training and inference, they can be implemented using hardwired logic and thus do not require memory circuits. Hence, RC is considered to be a promising alternative to replace DNNs, the model size of which continues to increase exponentially [31]. The state of each node is updated over M steps according to a nonlinear mapping F, which aims to map the low-dimensional input signal into the high-dimensional feature space as follows:

$$x_i(k) = F[\boldsymbol{x}(k-1), \boldsymbol{u}(k-1)]. \tag{2.1}$$

Here, $x_i(k)$ is the state of the i-th node at time step k, where $i \in \{1,2,\cdots,N\}$ and $k \in \{1,2,\cdots,M\}$. $\boldsymbol{x}(k) = \{x_1(k), x_2(k), \cdots, x_N(k)\}$ is the N-dimensional node state vector. Further, $\boldsymbol{u}(k) = \{u_1(k), u_2(k), \cdots, u_N(k)\}$ is the N-dimensional input vector, where $u_i(k)$ is the input to the i-th node at step k. As Eq. (2.1) shows, the state of node $x_i(k)$ depends on the previous state $x_i(k-1)$ and input $u_i(k-1)$.

Note that the weights in the reservoir, which represents the nonlinear mapping, can be fixed during training and inference. Hence, the reservoir can be implemented using hardwired logic, which contributes to a reduction in hardware resources. However, the nonlinear mapping still requires floating-point computations, which may lead to constraints when considering the implementation of RC in portable devices. To eliminate floating-point computations in the reservoir, several studies [34, 35, 62] have proposed exploiting cellular automata (CA) as an alternative components to traditional reservoirs, which are summarized in the following Subsection 2.1.2.

2.1.2 Suitable Components for Hardware Implementation

The key idea of RC is to may the input to a higher-dimensional space to facilitate the classification. As the basic components of the RC system, various methods have been proposed to perform the *reservoir* and *classifier*. The following techniques are commonly used because they are considered suitable for hardware implementation.

Cellular Automata

To reduce the massive usage of arithmetic units, the use of cellular automata (CA) has been proposed as a promising alternative to reservoirs [35]. A CA consists of multiple cells aligned in a one-dimensional array, where each cell takes two possible discrete states ("1" or "0") and evolves in discrete time steps. This evolution process is guided by specific rules and interactions between the nearest neighbors. With rich pattern dynamics, which makes CA very well suited to the hardware implementation of reservoir structures. The CA is a discrete computational model consisting of a regular grid of cells, each in one of a finite number of states. The state of an individual cell evolves in time according to a fixed rule, depending on the current state and the states of its neighbors. CA governed by certain rules have been proven to be computationally universal, that is, capable of simulating a Turing machine [63].

Ref. [34, 62] perform exhaustive studies of the performance of different CA rules when applied to pattern recognition of time-independent input signals using an RC scheme. Nichele *et al.* [62] evaluates the model on a 5-bit task, which is insufficient for the complicated application field. Moran *et al.* [34] selects the most accurate CA rules to represent the reservoir structure, which can easily be reproduced using a set of XOR gates and shift registers. They achieve a high-performance alternative for RC hardware implementation in terms of circuit area, power, and system accuracy, which can be considered as a low-cost method to implement fast pattern recognition digital circuits. However, this model exploits the softmax function as its classifier, which still requires FP calculations; hence, it is not suitable for implementation on resource-constrained devices. Lopez *et al.* [35] utilizes a random forest algorithm as the classifier, which avoids the costly FP calculation during the inference but also brings higher consumption for the training.

Bloom Filter

Bloom filters (BFs) are probabilistic data structures that represent a set as a small bit array allowing the occurrences of false positives, i.e., in a Bloom filter, an element can be incorrectly classified as a member of a set when it is not. Such memory-oriented classifiers for pattern recognition are typically very simple and can be easily implemented in hardware and software.

In [44], *Bloom WiSARD* was presented as an optimized framework that utilizes the BFs in a memory-segment way. Compared with the standard BFs, this model significantly reduces the memory requirement to some extent at the cost of allowing false positives and shows practically useful performance in image recognition tasks. The elimination of FP calculation and fast single-pass training are important advantages of *Bloom WiSARD* in terms of hardware efficiency. However, a large amount of memory required remains a key bottleneck. Take the MNIST classification task [64] as an example, when the false positive rate is 10%, the memory requirement is over 800MB for the inference implementation. It makes the adoption of *Bloom WiSARD* impractical for use in portable devices or memory-constrained systems.

2.2 Hyper-Dimensional Computing

2.2.1 Basic of Hyper-Dimensional Computing

HDC is a computing paradigm involving long vectors with dimensionality in the thousands, which are called hypervectors. In high-dimensional space, there are several nearly orthogonal hypervectors. HDC exploits well-defined vector operations to combine these hypervectors, while also preserving most of the information of the hypervectors [65]. Hypervectors are holographic and (pseudo) random with independent and identically distributed components as well as full holistic representation, thus no component has more responsibility to store any piece of information than any other hypervector.

Fig. 2.2 shows the overview of the classification in high dimensional space. HDC system generally consists of an encoder and an Associative Memory (AM). For all sample data within a class, HDC maps data to high dimensional vectors, called *hypervectors*, then combines them together to create a single hypervector modeling each class. Thereafter, the encoded hypervectors belonging to the same prediction class (label) are accumulated to build up the class's hypervector. All trained class hypervectors are stored in the AM. During the inference process, the same encoding scheme maps test input data to high dimensional space. AM looks at the similarity of the generated query hypervector against all stored class hypervectors. The input then gets the label of that class with which it has the highest similarity.

General Encoding Approach

The goal of HDC is to represent the input data in high-dimensional space. In the current HDC algorithms, models need to encode the input data to a single hypervector $\vec{H} = \langle h_1, ..., h_D \rangle$ with D dimensions as well as the corresponding precision. Here

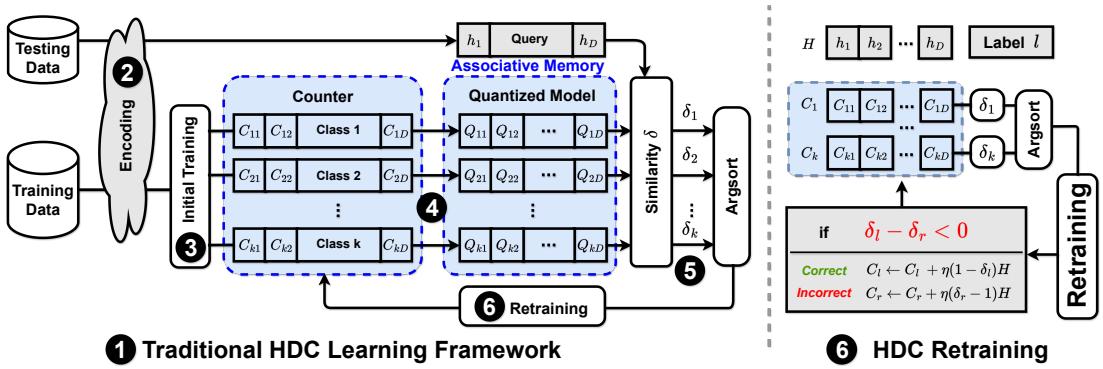

Figure 2.2: (a) Overview of General HDC. (b) Retraining Process.

the elements of hypervector $\vec{H}$ are m-bits length, while the values of h_i are in the range of $(-2^{m-1}, 2^{m-1})$. Consider an input data represented by the feature vector $\vec{F} = \langle f_1, f_2, ..., f_n \rangle$, where $n \ll D$ is the number of features for each input data. In Ref. [66], Aygun *et al.* summarized many encoding styles for the prior HDC. The difference of encoders epitomizes from two perspectives: (i) The operation in the high-dimensional space. *E.g.*, the binding in Record-based encoder, the bundling in Random-projection encoder, and the permutation in N-gram based encoder [42]. (ii) The way to consider the impact of each feature value on the final hypervector. For the reference, we explain the functionality of three popular encoders in detail:

Encoder I: Record-based encoder

Fig. 2.3 (a) shows the functionality of this encoding scheme, which was proposed and utilized in [24, 67]. Assume the original data point has n features $\{f_1, ..., f_n\}$. The first step is to quantize the range of pixel values into m levels. Then it assigns a random binary hypervector with D dimensions to each quantized level $\{\vec{L}_1, ..., \vec{L}_m\}$, where L_i is the i^{th} feature values level. The number of dimensions D in the hypervector is large enough compared to the number of features ($D >> n$) in the original data. The level hypervectors are generated such that the neighbor levels have higher similarity, as their absolute values have closer distance. To take the impact of each feature position under consideration, the encoding module assigns a random binary hypervector to each existing feature index $\{\vec{ID}_1, ..., \vec{ID}_n\}$, where $\vec{ID} \in \{0,1\}^D$. These $\vec{ID}$s are randomly generated such that all features will have orthogonal $\vec{ID}$s. The encoding can happen by linearly combining the feature values over different indices, where a hypervector corresponding to a feature index preserves the position of each feature value in a combined set:

$$\vec{H} = \vec{ID}_1 \oplus \vec{L}_1 + \vec{ID}_2 \oplus \vec{L}_2 + ... + \vec{ID}_n \oplus \vec{L}_n. \tag{2.2}$$

Here the $\vec{H}$ is the non-binary encoded hypervector, $\oplus$ denotes the XOR operation, and $\vec{L}_i$ is the binary hypervector corresponding to the i-th feature of vector $\vec{F}$. The binarization

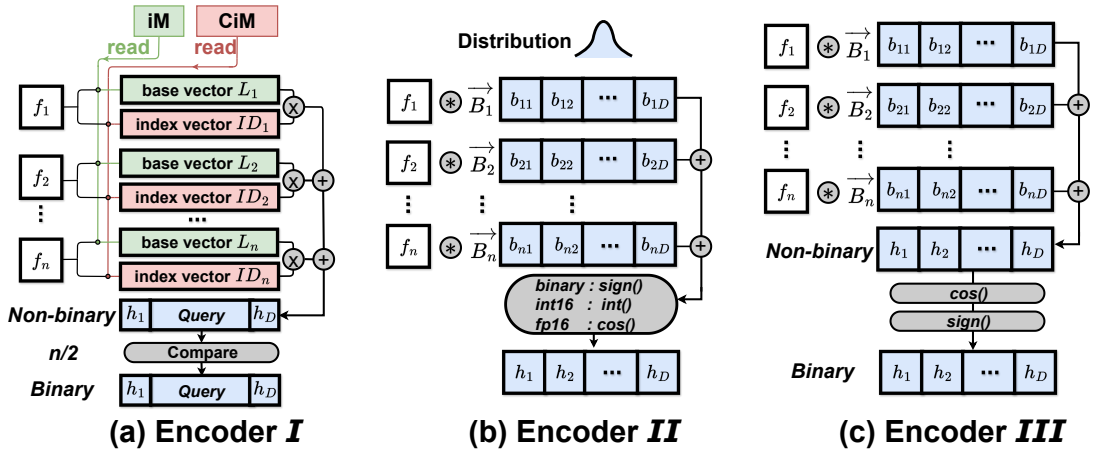

Figure 2.3: Functionality of three popular encoders.

of the encoded hypervector can happen by comparing each dimension of $\vec{H}$ with $n/2$ value. All dimensions with a smaller value than $n/2$ are assigned to 0, while other elements are assigned to 1.

Encoder II: Random-projection encoder

Unlike Encoder I, the second encoder differentiates feature positions by multiplying feature values with the corresponding index hypervector, $\vec{ID} \in \{-1,1\}^D$ and adding them for all the features. Fig. 2.3 (b) shows the functionality of the second encoding scheme when generating binary hypervectors, which is proposed in [37]. For example, where f_i is a feature value, the following equation represents the generation of encoded hypervector $\vec{H}$:

$$\vec{H} = sign(f_1 * \vec{ID}_1 + f_2 * \vec{ID}_2 + ... + f_n * \vec{ID}_n) \tag{2.3}$$

Here the $\vec{H}$ is the binary encoded hypervector, the $*$ denotes the element-wise multiplication, and *sign* is a sign function that maps the elements of results to '+1' or '0'.

Since the element in $\vec{ID}$s is '-1' or '+1', which simplified the element-wise multiplication of $f_i * \vec{ID}_i$ to the wire connection in hardware, *i.e.*, using the readout data from the index hypervectors $\vec{ID}$s as the sign bit of input features f_i. This encoder also assigns a unique index hypervector $\vec{ID}$ to each feature position and is stored in the Channel item Memory (CiM). But on the other hand, the item Memory (iM) is eliminated at the cost of broadening the range of summarizing results $\sum_{i=1}^{n} f_i * \vec{ID}_i$.

Note that such encoding scheme also capable of generating hypervectors with integer/floating-point elements. Similarly, for the generation of hypervectors with m-bits integral elements, the first step is to assign a random hypervector with D dimensions for each feature value, i.e., $\{\vec{B}_1, \vec{B}_2, ..., \vec{B}_n\}$, called base hypervector $\vec{B}_i$. For the HDC model with 16-bits integral or floating-point elements, we are randomly generating the base hypervectors $\vec{B}_i$ from a Gaussian distribution (mean equals to 0 and standard devia-

tion equals to 1) with the dimensionality of D. Since the base hypervectors are generated from the random bit streams, the similarity of different base hypervectors is nearly orthogonal. For each dimension, different features are combined by multiplying feature values with the corresponding base hypervector and addition.

$$\vec{H} = f_1 * \vec{B_1} + f_2 * \vec{B_2} + ... + f_n * \vec{B_n} \tag{2.4}$$

where $*$ represents the scalar multiplication and can make a hypervector that has integral elements. Similarly, for hypervectors with floating-point (FP) elements, different features are combined by multiplying feature values with the corresponding base hypervector, addition, and Cosine arithmetic.

$$\vec{H} = cos(\sum_i^n f_i * \vec{B_i}) \tag{2.5}$$

In this way, each input data can be mapped/encoded to the high dimensional space and represented by a single hypervector with D dimensions as well as the corresponding precision (fp16, int16, and binary). Note that this encoder is sensitive to data preprocessing. Normalizing or standardizing the features of input data before encoding can benefit the performance of the HDC model.

Encoder III: Non-linear encoder

Fig. 2.3 (c) shows the functionality of the third encoding scheme, which is proposed in [68]. This method explicitly considers non-linear interactions between input features. Though the data is not linearly separable in original dimensions, it might be linearly separable in higher dimensions. To generate a binary hypervector $\vec{H} = \{h_1, h_2, ..., h_D\}$ with D dimensions from an input feature vector $\vec{F} = \{f_1, f_2, ..., f_n\}$, the first step is to calculate a dot product of the feature vector with a randomly generated vector as $h_i = cos(\vec{ID_i} \cdot \vec{F})$. Here the $\vec{ID_i}$ represents the index hypervector, which is a randomly generated vector from a Gaussian distribution (mean λ=0 and standard deviation σ=1) with the same dimension as the feature vector. After this, the final encoded hypervector can be obtained by binarization with a sign function.

Note that to ensure the index hypervectors $\vec{ID}$ follow the Gaussian distribution and keep the summaries results in a suitable range before applying cosine function, the elements in $\vec{ID}$ need to be floating-point data, which seriously increases the memory requirement in the CiM. Meanwhile, in comparison with Encoder II, it requires the more expensive floating-point arithmetic, which limits the computation efficiency during the encoding procedure.

Table 2.1 shows the comparison between these three popular encoders. The values of level hypervectors $\vec{L}$ and index hypervectors $\vec{ID}$ are stored in the item Memory(iM) and the Channel item Memory (CiM) respectively. Take the Encoder I utilized in [24] as an example, for different classification tasks, the occupations of memory cost during

Table 2.1: Arithmetic Comparison of HDC Encoders in the Previous Works.

	Encoder I	**Encoder II**	**Encoder III**
CiM	✓	✓	✓
iM	✓	—	—
Arithmetic	XOR int Add sign	— int Add sign	FP Mul FP Add sign & cos

Table 2.2: Inference Memory Occupations in popular HDC model BinHD.

	MNIST [64]	**ISOLET** [69]	**UCIHAR** [70]	**FACE** [71]
CiM	94.92%	91.41%	92.73%	94.70%
iM	3.87%	4.74%	5.29%	4.98%
AM	1.21%	3.85%	1.98%	0.31%

the inference are shown in Table 2.2. We observed that the CiM and iM averagely take more than 93.4% and 4.7% of the inference memory cost, respectively. Meanwhile, the arithmetic difference is seriously affecting the hardware resource requirement, *e.g.*, the FP calculations in Encoder III increase the computational complexity and memory requirement under the same number of dimensions.

Initial Training

The simplicity of HDC training makes it distinguished from conventional learning algorithms. HDC training simply adds all hypervectors of the same class to generate the final model hypervector [39], which represents initial training Fig.2.2(❸). For example, after generating all encoded integer hypervector $\vec{H}^l$ of inputs belonging to the class l, the class hypervector $\mathbb{C}^l$ can be obtained by adding all $\vec{H}^l$ with the counter. Assuming there are $\mathcal{J}$ inputs having label l: $\mathbb{C}^l = \sum_j^{\mathcal{J}} \vec{H}_j^l$

Similarity Measurement

The first step of inference is to get the class hypervectors $\mathbb{Q}^j$ with corresponding precision from counter class hypervectors $\mathbb{C}^j$ as:

$$\mathbb{Q}^j = \{\mathbb{Q}_1^j, ..., \mathbb{Q}_D^j\} = Quant(\mathbb{C}^j) \tag{2.6}$$

where $Quant()$ is a quantization function that maps the elements of results to elements with m-bits precision. Such a quantization process Fig.2.2(❹) can effectively reduce the hardware cost for inference implementation. Then we create the encoded query hy-

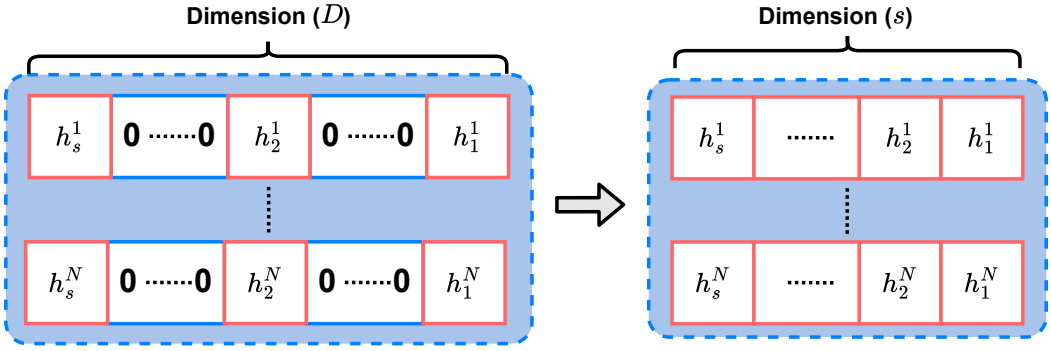

Figure 2.4: Dimension-wise Model Sparsification in HDC System.

pervectors $\vec{H} = \langle h_1, ..., h_D \rangle$ from the input using the encoding module described above. In this way, the HDC system with FP/integral elements can compute the Cosine similarity Fig.2.2(❺) of each encoded testing data with all the quantized class hypervectors $\mathbb{Q}^j$. For the HDC system with binary elements, we can simplify this process and utilize Hamming matrix for similarity measurement. In this way, the HDC system can compute the similarity Fig.2.2(❺) of each encoded testing data with all the quantized class hypervectors $\mathbb{Q}^j$. The class with the highest similarity is considered as the predicted label.

Model Sparsification

The goal of HDC at inference is to find a class hypervector with the highest similarity to the query hypervector, which is relative to the class hypervectors. However, not all dimensions of the class hypervectors have useful information that can distinguish one class from others. In some of the dimensions, all class hypervectors store common information shared among all classes, which add relatively similar weight to all classes in calculating the Cosine distance or Hamming distance. Ref. [40] proposed a framework to explore the sparsity of hypervectors on the class-wise and dimension-wise, which enables discarding the elements with minimal impact on the results and discarding the inconsequential (non-informative) dimensions shared across all learned hypervectors. The number of dimensions in hypervectors can be equally reduced and the classification accuracy can be improved with the same inference memory cost.

For the dimension-wise sparsity in HDC, the changes in the class elements in each dimension should be measured. After obtaining the variation of dimensions, the dimensions with the lowest change are selected to be dropped from the HDC model as they have the least impact on differentiating the classes. Fig. 2.4 shows the overview of sparsification on dimension-wise.

However, the redundancy of dimensions in hypervectors is not the only limitation.

The difference of contributions from each feature should be taken under consideration, especially in the image classification tasks. In most of the current HDC algorithms, all the pixels of images play an equal role in the generation of the non-binary hypervectors. Then it applies the majority voting to get the binary hypervectors as the representation of the input image. In this encoding mechanism, the crucial information from the key patterns could be minified by the noise from the other less important pixels, which may limit the application of HDC in image processing tasks. Hence, we proposed a novel sparsification mechanism to match our *StrideHD* model, which can be applied on the dimension-wise and feature-wise. The details will be described in Chapter 4.

Retraining

For the conventional retraining in Fig.2.2(❻), the encoded hypervector of each training sample is created as $\vec{H}$, and then the similarity with the j-th class hypervector $\delta_j = \delta(\vec{H}, \mathbb{Q}^j)$ is checked. Hence, δ_l represents the similarity for target label l. Taking δ_r as the highest similarity within all incorrect classes:

$$\delta_r = Max(\delta_{incorrect}). \tag{2.7}$$

When $\delta_l - \delta_r < 0$, the target label l is not considered as output, which means the prediction is mismatch to incorrect label r. In such case, the counter class hypervector $\mathbb{C}$ are updated as follows:

$$\begin{cases} \mathbb{C}^l = \mathbb{C}^l + \eta(1-\delta_l)\vec{H} \\ \mathbb{C}^r = \mathbb{C}^r + \eta(\delta_r - 1)\vec{H} \end{cases} \tag{2.8}$$

where η is a learning rate. In such iterative retraining, each encoded data is added to class hypervectors depending on how much new information the pattern adds to class hypervectors [39]. The retraining continues for multiple iterations until the validation accuracy has small changes and gets convergence during the last few iterations. Here the conventional model retraining only refers to the iterative learning for the misprediction samples, while we proposed a margin enhancement technique during the retraining phase. The details are shown in Chapter 5.

2.2.2 Related Works

Hyperdimensional Computing (HDC) leads to fast learning ability, high energy efficiency and acceptable accuracy in learning and classification tasks [72]. Besides, many researchers have been exploring the application of the high-dimensional computing paradigms on different datasets, which are shown in Table 2.3. For the further improvement of memory efficiency, classification accuracy, and robustness, many works have been proposed.

Table 2.3: Information about datasets covered by high-dimensional computing paradigms.

Dataset	Feature	Classes	Train Size	Test Size	Description
MNIST	784	10	60,000	10,000	Number Recognition
K-MNIST	784	10	60,000	10,000	Kuzushiji Recognition
F-MNIST	784	10	60,000	10,000	Fashion Recognition
ISOLET	617	26	6,238	1,550	Voice Recognition
UCIHAR	561	12	6,213	1,554	Activity Recognition (Mobile)
FACE	608	2	522,441	2,494	Face Recognition
PECAN	3112	3	22,290	5,574	Urban Electricity Prediction
PAMAP2	75	5	611,142	101,582	Activity Recognition (IMU)
APRI	36	2	67,017	1,241	Performance Identification
PDP	60	2	17,385	7,334	Power Demand Prediction

Accuracy & Memory Efficiency

The first general idea is to utilize hypervector with high-precision elements. In Ref. [36–41], authors need to encode data points to hypervectors with non-binary elements, *i.e.*, storing integer or FP value for each element. This leads to the high memory requirement and expensive computational cost. To reduce the inference cost, Rahimi *et al.* [42] and Imani *et al.* [24] propose binarizing the elements of class hypervectors after the training. Although these approaches can simplify the inference similarity matrix to Hamming distance and lead to a faster computation speed, it comes with the cost of significantly degraded HD classification accuracy on practical image recognition applications. For instance, Imani *et al.* [40] proposed a HDC model for face image classification task. If the model is binarized, the accuracy performance sharply decreases to 38.9%, which is far lower than the non-binarized mode.

To recover the model performance degraded through the quantization step, iterative training algorithms have been proposed [36–41,45]. After encoding all the training data point into high-dimensional binary vectors, these hypervectors are stored in the Associative Memory (AM). Utilizing the labeled training data, the similarity between the encoded hypervectors and stored hypervectors of each class can be measured. According to the correctness of prediction, the HD models are adjusted and optimized iteratively. By employing such gradient descent, the error rate of the HD model can be significantly reduced. However, this strategy requires tens of iterations to adjust the model, which leads to a long training time in comparison with the single-pass training.

Since the number of dimensions in the HDC model is strictly related to the performance of classification accuracy, Imani *et al.* [40] proposed a framework to sparse the

HDC model. They explore the prospect of sparsity in hypervectors to improve HDC efficiency without serious loss to accuracy. The research mainly focused on the sparsity of hypervectors class-wise and dimension-wise. Such strategy enables discarding the elements with minimal impact on the results and discarding the inconsequential (non-informative) dimensions shared across all learned hypervectors, which means the number of dimensions in hypervectors can be equally reduced. In this way, the classification accuracy can be improved with the same memory cost.

However, the redundancy of dimensions in hypervectors is not the only limitation. The difference of contributions from each feature should be taken under consideration, especially in the image classification tasks. Most of the current HDC algorithms use a simple encoder, in which all the pixels of images play an equal role in the generation of the non-binary hypervectors. Then it applies the majority voting to get the binary hypervectors as the representation of the input image. In this encoding mechanism, the crucial information from the key patterns could be minified by the noise from the other less important pixels, which may limit the application of HDC in image processing tasks.

Model Robustness

Regarding to the robustness of HDC system, Ref. [42,50] propose a robust HDC that can tolerate approximately 5×10^{-7} probability of failure of memory cells while maintaining accuracy. However, it is still far away from the solution to serious memory failure issues induced by scaling down the voltage to the sub-threshold region.

Poduval *et al.* [73] develops a learning system to include the traditional feature extractor into HD space. By operating accurate and robust learning over raw generated data, this work enables an entire learning application, including feature extractor algorithms like Convolution and Fast Fourier Transform, to process using HDC data representation. This model provides competitive robustness to the possible noise compared with the traditional HDC with feature extractors in the original space. They indicate that for the binary data representation in HD space, an error that only flips a reference dimension results in minor changes in the entire hypervector pattern. In contrast, an error for the feature extractor in the original space can happen in the most significant bits, which significantly affects the absolute value and robustness. However, this model mainly focuses on reducing the impact of hardware failure during the extra feature extraction phase.

Poduval *et al.* [74] proposes a data recovery mechanism as a runtime framework that adaptively identifies and regenerates the faulty dimensions unsupervised. This mechanism mainly focuses on handling the bit flip attack and random bit flip introduced by the temporary noise. By detecting the position of memory failures, it writes the correct values back to the memory cells statistically. However, this strategy only targets the

temporary bit-flip attacks instead of the permanent stuck-at errors. It assumes that the correct model and data can be written back to the attacked memory cells. When it comes to voltage-scaled circuits, permanent bit-stuck errors constitute a significant proportion of functional failures. Hence, an energy-efficient and robust HDC learning framework against both temporary and permanent memory errors is required.

Zhang *et al.* [75] explores two low-cost error masking techniques (word-level and bit-level) that can detect and mask errors. Utilizing a simple Razor circuit based on double sampling, this method can detect the existence and location of memory errors. Upon the detection of a bit flip, their scheme can mask the error by setting the faulty words or bits to logic 0s. After setting the corresponding words or most significant bits to 0s, the data bias induced by fault bits is limited, which reduces the impact of memory errors and improves the error resilience of the HDC models. Note that there is room for the native round-to-zero mechanism to be further improved. Also, such detection and masking process is executed during the inference, which results in the area and power overhead.

Alejandro *et al.* [39] proposes an HDC model with significant improvement in terms of robustness. The authors revealed the explanation of robustness in HDC systems. The logical value stored at each bit in the hypervectors has the same impact on the inference results. Therefore, even in a part of bits can not represent correct values, HDC systems have the potential to guarantee stable operation, which makes a hypervector robust against errors in its components as compared with the conventional data representation. This work indicated that using the precision of elements in hypervector has an impact on the dimensionality and robustness of the HDC model. On one hand, the HDC model utilizing binary elements in Ref. [24,39] can obtain better robustness. On the other hand, the HDC model with more precise elements, e.g., 16-bit integer, can provide acceptable accuracy with fewer dimensions and reduce over 70% Energy-Delay Product (EDP) on CPU platform [39].

The emerging memory devices have various reliability issues such as endurance, durability, and variability. This, coupled with the high computational complexity of learning algorithms, results in many writes to memory resulting in endurance issues in the accelerators. Embedded devices are resource constrained. Instead of accelerating the existing algorithms on the embedded devices, we need to think how to design algorithms that mimic the efficiency and robustness of the human brain. To achieve real-time performance with high energy efficiency, we need to rethink not only how we accelerate machine learning algorithms in hardware, but also we need to redesign the algorithms themselves using strategies that more closely model the ultimate efficient learning machine: the human brain.

Hyperdimensional computing (HDC) is a strategy developed by computational neuroscientists as a model the human short-term memory. HDC is motivated by the under-

standing that the human brain operates on high dimensional representations of data originated from the large size of brain circuits. It models the human memory using points of a high-dimensional space, called *hypervectors*. The hyperspace typically refers to tens of thousand dimensions. These points can be manipulated with a formal algebra operations to represent semantic relationships between objects. HDC mimics several desirable properties of the human brain, including: robustness to noise and hardware failure and single-pass learning where training happens in one-shot without storing the training data points or using complex gradient-based algorithms.

Hyperdimensional (HD) computing is well suited to address learning tasks for IoT systems as: (i) HD models are computationally efficient (highly parallel at heart) to train and amenable to hardware level optimization, (ii) HD models offer an intuitive and human-interpretable model, (iii) it offers a complete computational paradigm that can be applied to cognitive as well as learning problems, (iv) it provides strong robustness to noise, which is a key strength for IoT systems, (v) HD can naturally enable secure and lightweight learning. These features make HD computing a promising solution for today's embedded devices with limited storage, battery, and resources, as well as future computing systems in deep nano-scaled technology which devices will have high noise and variability.

2.3 Voltage Scaling for Low Power Design

Energy consumption in circuits largely results from the charging and discharging of internal node capacitances and can be reduced quadratically by lowering supply voltage (V_{DD}) [76]. In [46], compared with the nominal voltage operation, they achieved 4.7× energy consumption reduction by scaling down the voltage to the near-threshold voltage region. For the application of voltage-scaled circuits, the portable medical devices (*e.g.*, blood pressure monitors, glucose monitoring systems, automated insulin pumps, ECG monitors) can be the suitable scenarios. Such portable devices are expected to run in a long time, which requires extremely high energy efficiency. The patient information also needs to be protected due to the privacy concerns. Meanwhile, some the monitors needs to receive the bio-information under the unreliable and noisy situation. If we further downscale the supply voltage, the minimum energy voltage can be found in the sub-threshold region [47]. However, low-voltage designs may suffer from functional failures due to various causes, e.g., soft errors, aging, and processing variation.

Static Random Access Memories (SRAMs) are one of the most vulnerable circuits to such variations. For example, if we consider process variation in SRAMs, some bit cells can not be accessed correctly. For 65nm SRAM cells at the nominal voltage which has a typical failure probability of $\sim 10^{-7}$ increases to approximately 4% at the voltage of 500mV due to process variations [55, 56]. If we target to further downscale the V_{dd}

in advanced process technologies such as [77], the failure probability sharply increases. Ref. [78] shows the Monte Carlo simulation results of 6T SRAM bit cells with a 65-nm process technology in a low voltage region. If we consider the FS corner case, the read-out failure probability sharply increases in the low-voltage region. The results indicate that the readout failure probability reaches more than 10% with a 500 mV supply voltage, which becomes the fundamental drawback of the application in the sub-threshold voltage region. One possible solution is to design memory blocks that are robust to such variation at the cost of increasing the hardware resource of VLSI circuits [46], which is not desirable for resource-constrained edge-oriented devices.

Several recent works studied the solution to circuit failure or bit-flip attacks in memory, mainly targeting machine learning applications. When it comes to the DNN models, which usually have high sensitivity to noise and often require floating-point precision for calculation, this magnitude of circuit failure probability induced by voltage scaling is unbearable. Though [79] proposes a novel reconfigurable SRAM architecture for undermining the impact of memory failure in higher-order bits, changing the well-optimized memory block could be significantly costly and less desirable for the industry. Under a 10% memory failure rate, approximately 68%, 36%, 19%, and 3.1% accuracy loss are suffered for DNN, support vector machines (SVM) [80], AdaBoost [81], and the state-of-the-art HDC model, respectively [39, 73, 74]. Hence in comparison with the traditional DNN models, the HDC models show great potential and the advantage of robustness in the application of edge-oriented voltage-scaled circuits.

2.4 Summary

Section 2.1 and Section 2.2 introduces the reservoir computing (RC) system and hyper-dimensional computing system, while both of these two computing paradigms aim to map the input data into high-dimensional space and can be considered as the energy-efficient alternatives. Section 2.3 shows the basics of voltage scaling, which is a classic and promising low-power design methodology. Finally, a summary of these techniques is presented in Section 2.4.

Chapter 3

RC System using CA and Ensemble BF

3.1 Introduction

With the increasing scale of deep neural networks (DNNs), most portable and wearable devices are becoming unable to handle the large memory consumption and computing demand in both the training and inference phases. For example, AlexNet [5] requires 249MB of inference memory and performs 1.5 billion high precision operations to classify one image. Even applying the hardware-friendly implementation techniques to get the Binarized Neural Networks (BNNs) [7] or XNOR-Networks [8], still require expensive computation costs due to the floating-point calculation and backpropagation algorithm during the training. Most of the small edge devices do not have sufficient computing power to accomplish such sophisticated algorithms. Hence, it is crucial to meet the rising demand for more computationally efficient models.

Reservoir computing (RC) is a promising alternative for drastically reducing the computational burden of machine learning methods. The most critical advantage of RC is that only some of the parameters are trained, while the rest can be fixed. Owing to this unique feature, RC can be implemented with limited hardware resources, that is, fixed weights can be realized using hardwired logic. The standard RC architecture generally consists of a *reservoir* and *classifier*. All the input signals are given to a reservoir, which is often constructed by a recurrent neural network (RNN) whose synaptic weights are randomly initialized [30]. After being fed into the fixed and nonlinear pattern dynamic reservoir, these input signals are mapped into a higher-dimensional feature space. Finally, the output of RC is obtained using the trainable linear layer.

In comparison with the conventional CNNs with hardware implementation techniques e.g., BNNs and XOR Net, the design strategy of RC systems effectively avoid the use of complex training method in the reservoir part, and thus the learning process is simplified to a classical regression problem. Because of its simplicity and low computational cost in both the training and inference phases, RC systems have been success-

fully applied in many different fields, such as image recognition and robot control [31]. However, the frequent use of floating-point (FP) arithmetic found in most existing RNN models makes the implementation of RC systems on hardware challenging.

To reduce the massive usage of arithmetic units, the use of cellular automata (CA) has been proposed as a promising alternative to reservoirs [35]. A CA consists of multiple cells aligned in a one-dimensional array, where each cell takes two possible discrete states ("1" or "0") and evolves in discrete time steps. This evolution process is guided by specific rules and interactions between the nearest neighbors. With rich pattern dynamics, CA is very well suited to the hardware implementation of reservoir structures.

The Bloom filter (BF) is a space-efficient probabilistic data structure aimed at approximate member queries, that is, testing whether an element belongs to a given set [82]. This filter can be treated as a special case application of HD computing models. By adopting the BF, a weightless neural network has been proposed for image classification tasks, successfully eliminating costly FP calculations while maintaining fast single-pass training and yielding satisfactory accuracy performance [43,44]. In contrast to conventional neural networks, the BF-based model is characterized by its simple implementation in both software and hardware. However, the large amount of memory required remains as the bottleneck for the application of BF, which makes it impractical when used for a portable device or hardware-resource-constrained system.

To further reduce the memory footprint, we propose a novel RC model: *EnsembleBloomCA*. Similar to the *ReCA* proposed in [34], *EnsembleBloomCA* adopts CA as the reservoir. The uniqueness of *EnsembleBloomCA* lies in the utilization of the ensemble Bloom filter as a classifier, which can alleviate the pollution of Bloom filters, even when memory capacity is limited, and thus contribute to the significant reduction of memory cost. The main contributions of our model are the following:

- Eliminating all floating-point calculation and integer multiplication, which makes *EnsembleBloomCA* suitable for hardware implementation.
- Achieving 43× memory reduction during inference in comparison with [43] without hurting the accuracy.
- Achieving reductions of over 23× and 8.5× in area and power consumption, respectively.

3.2 Preliminary

3.2.1 Cellular Automata (CA)

Because CA provides a simple method to map the input into a high-dimensional space, it is useful as a hardware reservoir. Therefore, we adopt CA as a reservoir in the proposed method, similar to the existing methods [34]. Elementary cellular automata (ECA) is

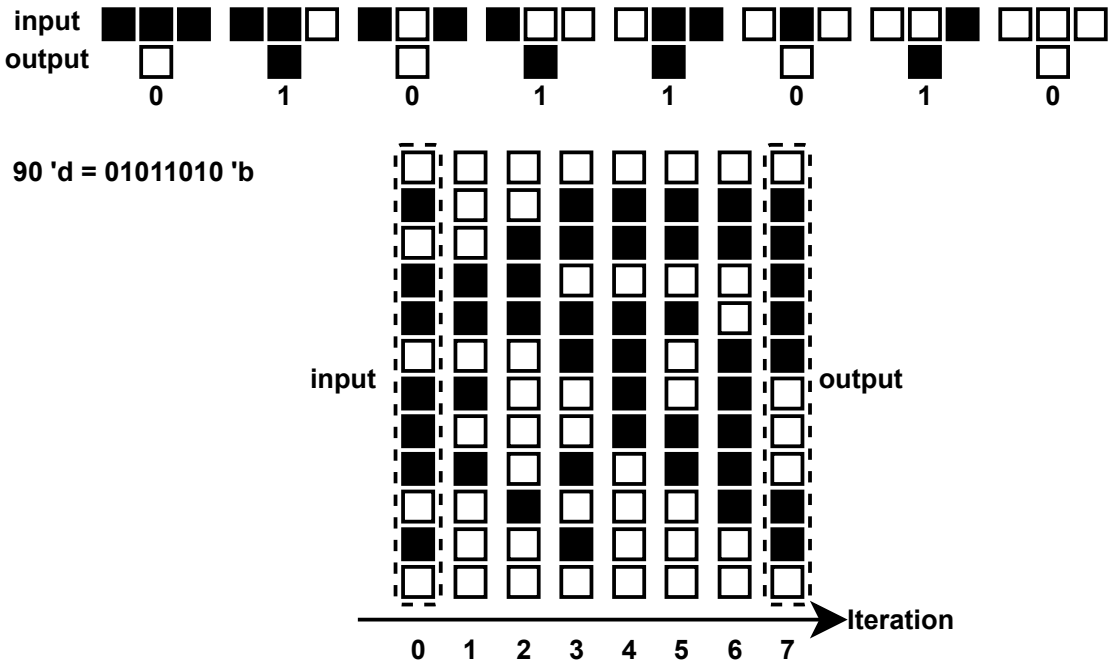

Figure 3.1: Example of cellular automata evolution in Rule 90.

the simplest class of 1-dimensional CA [83], where each cell takes binary states, *i.e.*, either "1" or "0." The updated state of a cell is determined by three cells, *i.e.*, the cell and two neighboring cells, and hence, the time evolution of cell states can be written as

$$x_i(k) = F[x_{i-1}(k-1), x_i(k-1), x_{i+1}(k-1)]. \tag{3.1}$$

There are $2^{2^3} = 256$ possible evolution rules in total which can be labeled from Rule 0 to Rule 255. Fig. 3.1 is an example of ECA Rule 90.

3.2.2 Cellular Automata Applied to Reservoir Computing

For the image classification tasks, assume the image u as the input data of the RC system. To apply CA to RC, the internal nodes and inputs should be converted into a binary format. Ref. [34] proposed the use of thermometer encoding, where an n-bit integer value is converted into a 2^n-bit binary string as follows:

$$u^{(l)} = \begin{cases} 1, \frac{R \cdot l}{d} < u, \\ 0, \frac{R \cdot l}{d} \geq u, \end{cases} \tag{3.2}$$

where d is the length of the binary data, R is the range of intensity for each pixel, u is the original input image with decimal data, and $u^{(l)}$ represents the l-th channel of the binary input data. For each pixel, the vector is initialized with d bits of 0s. The bits ranging from the most significant bit (MSB) to the first bit with a threshold higher than the pixel value are changed to 1. All the integer values $u \in [0,R]$. Thus, the thresholds of thermometer encoding can be obtained by dividing the pixel space R into d parts.

This encoding mechanism has been proven to significantly increase the error tolerance of neural networks, especially in terms of constructing adversarial samples [84]. Meanwhile, we note that there is another advantage of our model for hardware implementation, which benefits from thermometer encoding and is rarely noticed. This binary encoding mechanism can implement the max-pooling function utilizing only bitwise OR gates. This hardware friendliness decreases the energy consumption for both the training and inference phases.

After obtaining the binarized input signal, the ECA rule is applied to rows and columns independently with a fixed boundary condition. These two image results are combined with a bitwise XOR operation. This process is repeated for M iterations; for all images, rows and columns are independently iterated over with the same ECA rule, and the resulting vectors are combined with a bitwise XOR operation.

After decomposing the original images into d binary channels, we obtain the l-th channel of the input signal $u^{(l)}$, where $l \in [1,d]$. There is no communication between binary channels. The same ECA rule is repeated M times in total.

Let g^k be the function g applied k times, and let $x^{(l)}(k)$ be a Boolean time-dependent image, which can be expressed as

$$x^{(l)}(k) = g^k(u^{(l)}) = \begin{cases} u^{(l)}, & k = 0, \\ g^1(x^{(l)}(k-1)), & k > 0. \end{cases} \tag{3.3}$$

We can obtain the state of the reservoir in the i-th position, k-th iteration, and l-th binary channel as $x_i^{(l)}(k)$, where $i \in \{1,2,...,N\}$, $k \in \{1,2,...,M\}$ and $l \in \{1,2,...,d\}$. Thus,

$$x^{(l)}(k) = [x_1^{(l)}(k), x_2^{(l)}(k), ..., x_N^{(l)}(k)]. \tag{3.4}$$

The images are iterated over independently by rows and columns. We define $x_{row}^{(l)}(k)$ as the result of iterating images by rows, that is, the state of each updated cell is determined by two horizontal neighboring cells and the cell itself. Similarly, $x_{col}^{(l)}(k)$ represents the results of iterating over images by columns. The vectors $x_{feature}^{(l)}(k)$ are obtained by combining $x_{row}^{(l)}(k)$ and $x_{col}^{(l)}(k)$ with an XOR operation:

$$x_{feature}^{(l)}(k) = x_{row}^{(l)}(k) \oplus x_{col}^{(l)}(k). \tag{3.5}$$

The $x_{feature}(k)$ is defined as

$$x_{feature}(k) = \sum_{l=0}^{d-1} x_{feature}^{(l)}(k), \tag{3.6}$$

where $k \in [0, M]$. Subsequently, we apply a max-pooling layer to improve the generalization of the network and reduce the weights in the classifier. Because the internal states are binarized, the CA is suitable for digital hardware implementation. However, the classifier still requires a softmax operation in [34], which should be eliminated for ease of implementation on resource-constrained devices.

3.2.3 Application of Bloom Filter

To completely eliminate the FP calculations, we propose the exploitation of the Bloom filter (BF) to construct the classifier in the RC system. The Bloom filter is considered as a space-efficient probabilistic data structure, which aims to test whether an unknown item is a member of the given set [82]. As the term "probabilistic" suggests, the query result may contain errors, *i.e.*, the Bloom filter returns either "possibly in the set" or "definitely not in the set." From a neural processing point of view, BFs are a special case of an artificial neural network with two layers (input and output), where each position in a filter is implemented as a binary neuron. Such a network does not have interneuronal connections; that is, output neurons (positions of the filter) have only individual connections with themselves and the corresponding input neurons. The most significant advantage of the Bloom filter is its memory space efficiency over other data structures, which is suitable for error-resilient applications such as machine learning [85].

The standard BF allows the addition of new elements to the filter and is characterized by a perfect true positive rate (i.e., 1), but a nonzero false positive rate. The false positive rate depends on the number of elements to be stored in the filter, as well as the filter parameters, including the number of hash functions and the size of the filter. In the BF model, the element is considered as an L-bit binarized vector, which represents the position within the Bloom filter. Thus, each Bloom filter contains 2^L bits. We define $B(index)$ as the $index$-th bit in the Bloom filter, where $index \in [0, 2^L - 1]$.

In the insertion phase, all the values in the Bloom filters are initially set to zero. Each training sample is inserted into the corresponding Bloom filter based on their labels. The value of the accessed bit $B(index)$ is set to one. In the query phase, the testing sample is sent to all the Bloom filters and returns the value of the accessed bit $B(index)$ in every Bloom filter. When it returns positive (logic "1"), the testing sample is considered as "possibly in this category." When it returns negative (logic "0"), the sample is judged as "definitely not in this category."

Fig. 3.2 shows an example of Bloom filter operations with a 16-bit array and three

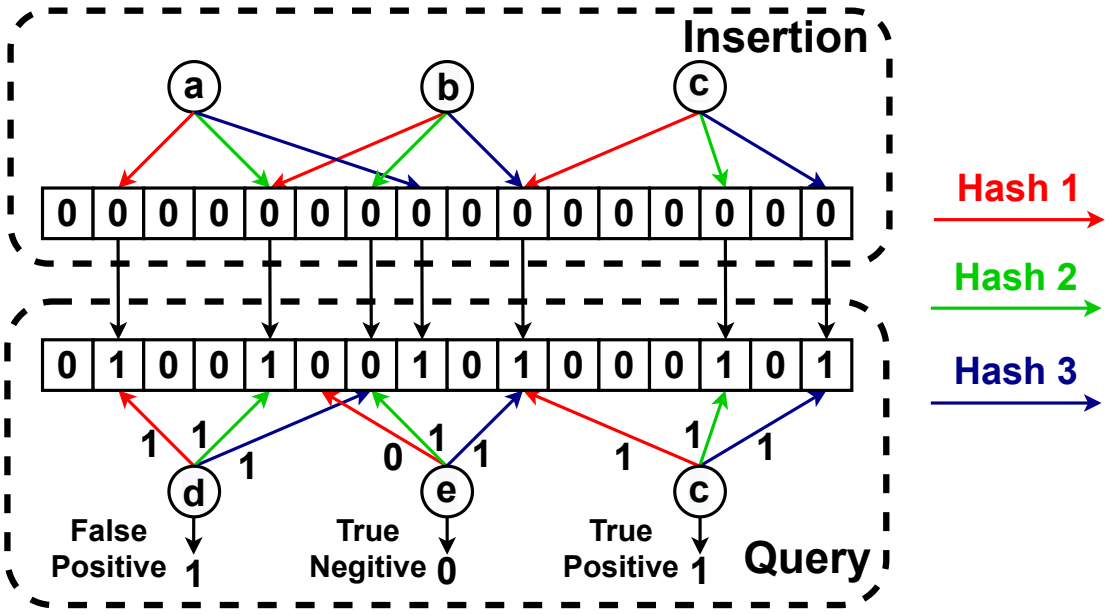

Figure 3.2: Example of Bloom filter operations with a 16-bit array and three hash functions.

hash functions. In the insertion operation, each element is mapped into three positions according to the three different hash functions (*e.g.*, MurmurHash [86]). Then these corresponding hashed bits are set to 1. The query operation looks up the positions mapped from the input element, indicating whether it is a member of the set. As Fig. 3.2 shows, d is a false positive, as it was returned as a member of the set (only a, b and c were inserted).

3.3 Proposed Method

3.3.1 *EnsembleBloomCA*

EnsembleBloomCA is a novel RC architecture comprising an ingenious combination of CA and an ensemble Bloom filter. Fig. 3.3 shows the overview of *EnsembleBloomCA*, which consists of CA for extracting a high-dimensional binary feature vector from an input figure, and Bloom filters, each of which corresponds to a class label. Once an input image is provided, *EnsembleBloomCA* first extracts the binarized feature vector in the same manner as in Subsection 3.2.1. Then, the similarity between the extracted feature vector and each Bloom filter is computed and the class whose corresponding Bloom filter exhibits the maximum similarity is output as the model prediction. In the following, we detail the algorithm of *EnsembleBloomCA*, which exploits the benefits of

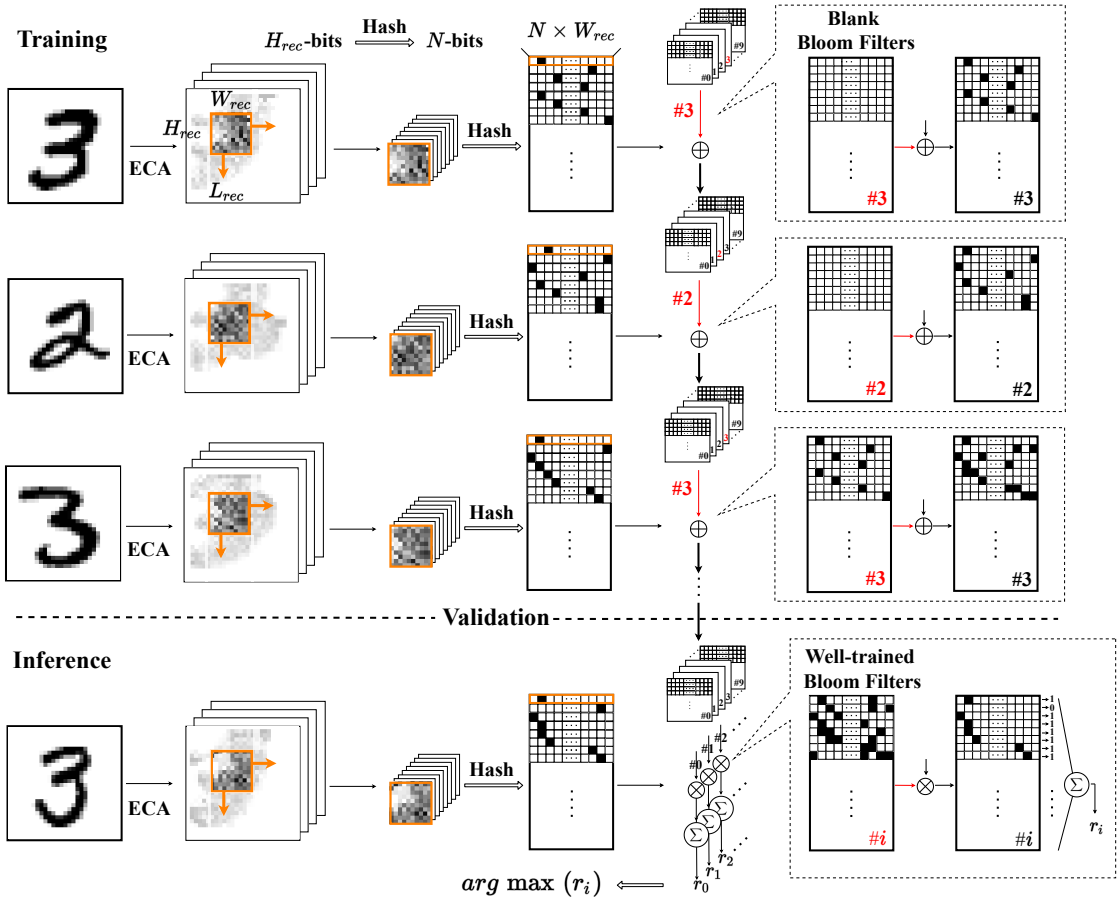

Figure 3.3: Overview of the proposed *EnsembleBloomCA*.

CA and Bloom filters.

Training Phase: Initially, the Bloom filter classifier is blank, that is, all the values are set to zero (logic "0"). Then, the input image is fed into the CA part for feature extraction. Here, the evolution rule is applied M times to elevate the pattern dynamics that are ready for classification. In the next step, we extract patches of the CA output by applying a $W_{rec} \times H_{rec}$ size receptive field with a stride of L_{rec} and again apply a simple hash function to each extracted patch to obtain a binary feature vector. Details regarding the hash function are provided in Subsection 3.3.2. Finally, the extracted feature vector is fed into the Bloom filter, which is specified by the corresponding training label, with bitwise OR gates, and the training for this image is completed. For example, a training image labeled "5" is inserted into Bloom filter #5.

Inference Phase: Similar to the training phase, the input images are fed into the CA, followed by image patch extraction and application of the hash function to obtain the feature vector for the input image. Then, using counters and bitwise AND gates to

calculate the similarity between the feature vector and Bloom filters, the Bloom filter with the highest response is chosen as a representative category for the testing image.

As can be seen, the similarity should reach the maximum value if the input pattern belongs to the corresponding category, which enables us to determine which class the unseen input pattern belongs to without using computationally expensive floating-point arithmetic. Although the pattern dynamics are extremely elevated by the CA reservoir, this also increases the number of Bloom filters in every category, which results in untenable memory usage during the inference phase. N_{sub}, which represents the number of samples inserted into each Bloom filter, has a significant correlation with the performance of the Bloom filter. The more elements are added to a single Bloom filter, the higher the probability of false positives (FPR) [87]. To optimize this data structure, we propose utilizing an ensemble Bloom filter as the classifier in our RC system.

3.3.2 Ensemble Bloom Filter

Ensemble learning is a machine learning paradigm in which multiple base learners are trained to solve the same problem. The generalization ability of an ensemble is usually much stronger than that of the base learners. The base learners are usually generated from training data using a base learning algorithm that can be a decision tree, neural network, or other type of machine learning algorithm [88].

Several algorithms are commonly used for ensemble learning, including bagging. In this method, training data subsets are randomly drawn from the entire training dataset. Each training data subset was used to train a classifier of the same type. Individual classifiers are then combined by taking a simple majority voting in the classifier or using relatively weak classifiers (such as decision stumps, an approach which constitutes a random forest classifier). Another popular method, boosting, also creates an ensemble classifier by resampling the data and then combining it through majority voting. However, in boosting, resampling is strategically geared to provide the most informative training data for each consecutive classifier [89].

In our case, we apply the bagging algorithm. The training dataset is divided into N_{sub} subsets and inserted into different Bloom filters as base learners. Then, the results of these base learners are summed up together, a process which can be considered as an ensemble Bloom filter classifier. The basic operations of the ensemble Bloom filter model involve adding elements to the corresponding set (insertion phase) and querying for element membership in the probabilistic set representation (query phase).

For the ensemble Bloom filter, a binarized pattern with $N \cdot L$ bits is split into N vectors of L bits. We define $B_n(i)$ as the i-th bit in the n-th mini Bloom filter, where $n \in \{1, 2, ..., N\}$. In our case, the patterns from the $W_{rec} \times H_{rec}$ size receptive field need to be memorized by a different ensemble Bloom filter. Similar to [43], each binarized

pattern is split into W_{rec} and H_{rec} vectors in rows and columns.

During the insertion phase, the vectors are inserted into the corresponding Bloom filters, which we call "mini Bloom filters." These vectors are considered as addresses within the mini Bloom filters, as well as the results of the hash function. In the query phase, if and only if all the values of accessed bit $B_n(i)$ in the mini Bloom filters are positive (logic "1"), this Bloom filter returns one. Otherwise, this Bloom filter returns zero. All the returned results of Bloom filters in each category are summed together and considered as the discriminator response r, which also represents the similarity between the testing sample and the corresponding category. The discriminator with the highest response r is chosen as the representative category.

After training each Bloom filter, we select those exhibiting good classification accuracy using validation samples. Similar to the training phase, the class labels of the validation images are predicted to evaluate the classification accuracy of each Bloom filter. Then, we select N_{inf} Bloom filters exhibiting the top N_{inf} performance rankings; these filters construct the classifier used in the inference phase.

The key idea behind using the Bloom filter as a classifier is to store the pattern information within the given set, which means that an excessive difference between patterns leads to the pollution of Bloom filters. Compared to the baseline model in Ref. [43,44], one of the main differences comes from the ensemble learning, which treats the Bloom filters as base learners. Instead of using a single standard Bloom filter with high memory cost, utilizing Bloom filters in an ensemble way can not only prevent the pollution from large scale of training data but also effectively improve the performance of image recognition tasks. Meanwhile, the number of well-trained Bloom filters in each Bloom filter pool decreases, which leads to a significant reduction in memory cost during the inference phase

3.4 Experiment

3.4.1 Experiment Setup

In our experiment, we focused on the handwritten digit number classification task based on the MNIST dataset, which is a collection of 70k handwritten digits in grayscale format. This task is extensively used to compare the performance of many classification models by evaluating the performance of a machine learning algorithm [64]. Among 60k images, we randomly selected 55k images for training and 5k images for validation. The training images were used to populate Bloom filters, while the validation images were used to optimize the hyperparameters, such as the CA evolution rules or evolution times. The remaining 10k images were used for testing. We will make a comparison between the following three methods:

Bloom WiSARD is the baseline algorithm in [43]. It is an optimized application of standard BFs, which utilizes BFs in a memory-segment way. The input images belonging to the same category with the size of 28×28, are split into 28 rows, and individually stored into the 28 BFs during the training. When it comes to the inference phase, each BFs returns the query results as logic "0" or "1". The sum of these query results represents the response of the corresponding category. Hence, the class with the highest response is considered as the output of the classification.

Ensemble Bloom filter is a special case of our proposed method. For the comprehensive exploration, we also evaluate the performance of our model without utilizing CA. In this case, no reservoir architecture is applied and we directly use the proposed ensemble Bloom filter in subsection 3.3.2 as the classifier, which can help us to evaluate the contribution of CA and ensemble Bloom filter individually.

EnsembleBloomCA is our proposed method. 55k training images were inserted into the ensemble Bloom filter as the training set. We then divided 55k training images into N_{sub} subsets for every category in order. The other 5k training images were used as the validation set for adjusting the ensemble Bloom filter classifier. Then, we utilized 10k testing images as the inference set to evaluate the performance of our approach.

In our experiment, $w = 28$, $h = 28$, $d = 16$, and $R = 256$. For the reservoir, CA maps the original input into a higher-dimensional space and obtains high-dimensional patterns. The max-pooling layer was selected to have a stride of two, a squared window of size two, and zero padding. A 5×5 receptive field was applied to every binary channel and iteration of the reservoir output, with a stride of 3. The software implementation was emulated in *C++* to evaluate the performance of *EnsembleBloomCA* .

3.4.2 Optimization of ReCA

Although the reservoir using CA recreates a rich pattern for the *EnsembleBloomCA* model, extra features also require more memory resources and data transfers. In [34], an approach using only the 8th iteration to train the classifier was proposed. Every time the iterative patterns are obtained with a fixed CA evolution rule, the pattern is changed to some extent.

Considering the difference in contribution from every iterative pattern, we only choose the first iterative pattern which represents the original input data and the latest iterative pattern as features. This strategy can effectively refine the input feature for the ensemble Bloom filter and reduce the memory cost in both the training and inference phases.

3.4.3 Experiment Results

Using the *EnsembleBloomCA* model described in section 3.3, we examined the performance of all existing CA evolution rules. The iteration count M is in the range from 1 to 24. As shown in Fig.3.4, the RC system achieves different accuracy performances under different ECA rules. According to the hyperparameter optimization from the validation set, we adopt the ECA evolution Rule 184 and iteration count $M = 8$.

Fig. 3.5 shows the classification accuracy as a function of the number of samples inserted into a single Bloom filter and memory cost required during training. The dashed horizontal line shows the classification accuracy of [43], which is the baseline of our work. When the training memory cost is lower than 400 KB, the accuracy drops sharply. In these cases, an excessive number of feature vectors are inserted into the same Bloom filter, which pollutes the Bloom filter and affects the accuracy of the ensemble Bloom filter. When the training memory cost is up to 825 KB, the accuracy trend appears to be saturated.

Table 3.1 shows the results of several different models in the MNIST handwritten number classification task. The *EnsembleBloomCA* model achieved over $819.05/18.75 \approx 43\times$ memory reduction compared with the baseline while maintaining the same accuracy. This reduction in memory cost mainly comes from the use of Bloom filters in an ensemble. Although this method also leads to a slight decrease in accuracy, this disadvantage can be overcome by using CA as the reservoir. The rich pattern dynamics recreated by CA can effectively provide more candidates for Bloom filter pools and improve accuracy from 89.58% to 91.86%, which also illustrates the impact of CA in our model.

For the *Bloom WiSARD* [43]architecture that splits the input images in row and stores the binary patterns with standard Bloom filters individually, its feature extraction procedure is homogeneous and lacks of focus on the regional information in the image processing tasks. On the other hand, the *reservoir* of our proposed *EnsembleBloomCA* utilizes the cellular automata to elevate the pattern dynamics, then further applies the techniques like receptive field and max-pooling to extract the information from input

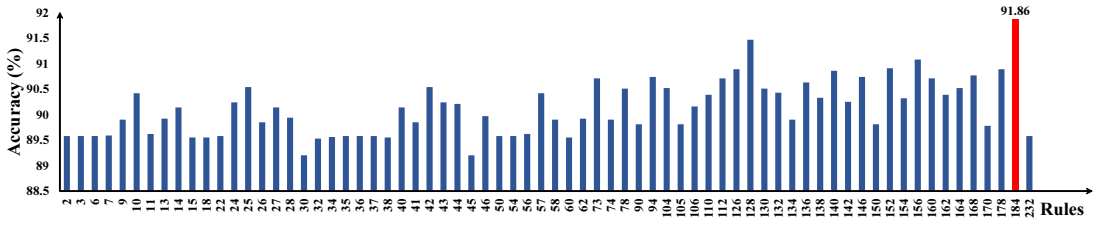

Figure 3.4: Accuracy Performance applying different ECA rules.

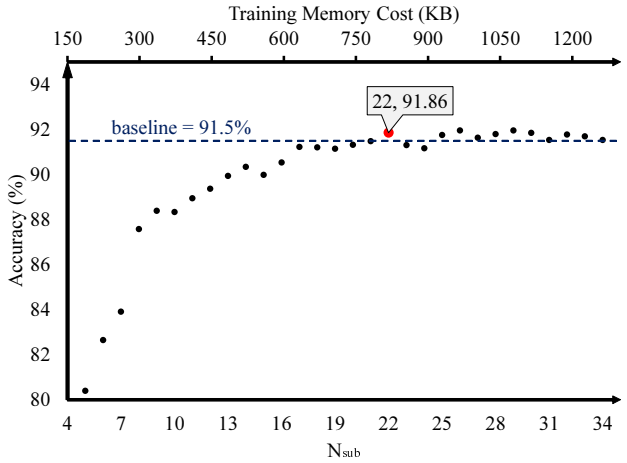

Figure 3.5: Accuracy - N_{sub} / Training Memory Cost.

Table 3.1: Performance Comparison.

	Bloom WiSARD (baseline)	**Ensemble Bloom filter (proposed)**	***EnsembleBloomCA* (proposed)**
Arithmetic	multiplication	addition	addition
Number of hash	3	1	1
Accuracy(%)	91.50	89.58	91.86
Inference Memory (KB)	819.05	18.75	18.75

images and map the input into high-dimensional space. Hence, the rich pattern dynamics recreated by CA can effectively provide more candidates for Bloom filter pools and improve the accuracy performance. Overall, our proposed model can significantly reduce the memory cost for the inference phase while maintaining the accuracy. Meanwhile, the simplicity of the hash function avoids high computational costs, allowing for practical hardware implementation.

3.4.4 Hardware Implementation

The hardware architecture of the baseline *Bloom WiSARD* and our proposed *EnsembleBloomCA* were designed using SystemVerilog. We used *Synopsys Design Compiler* to synthesize and report the area and power consumption of our approach in a 65-nm ASIC flow. The hardware costs of the memory part were individually simulated using *CACTI*, which is an integrated memory access time, area, leakage, and power model. In

Table 3.2: Hardware Performance Comparison.

	Area (mm^2)			**Power consumption** [mW]			**MPD**
	Reservoir	**Memory**	**Total**	**Reservoir**	**Memory**	**Total**	[$10^{-9}s$]
Baseline	0.206	7.805	8.012	258.9	1059.8	1318.7	33.77
Proposed	0.095	0.253	0.348	121.4	34.5	155.9	3.78
Reduction	2.2×	30.8×	23.0×	2.13×	30.7×	8.5×	8.9×

addition, the memory type was chosen to be the main memory in the 65-nm ASIC flow, which does not contain any tag array, and every access occurs at a page granularity. Table 4.3 shows the comparison between *EnsembleBloomCA* and [43] in terms of ASIC area and energy consumption. The maximum propagation delay (MPD) of our model is 3.78 ns.

The memory takes $0.253/0.348 \approx 72.7\%$ of the area of *EnsembleBloomCA*, while the other circuits take 37.3%. In terms of power consumption, the memory portion only uses $34.5/155.9 \approx 22.13\%$, and the power consumption percentage of other circuits is increased to 77.87%. Although the memory occupies almost half of the entire circuit area, the energy is mainly consumed by the non-memory part, that is, CA, owing to the high switching activity of the non-memory part. Overall, *EnsembleBloomCA* achieved over 23× and 8.5× reductions in area and power, respectively.

3.5 Conclusion

In this work, we propose a novel RC architecture, the *EnsembleBloomCA* model, which is a combination of a reservoir using CA and an ensemble Bloom filter classifier. By utilizing *EnsembleBloomCA*, we achieved a 43× reduction in memory cost for the inference phase while maintaining accuracy. Our hardware implementation also demonstrated that *EnsembleBloomCA* achieves over 23× and 8.5× reductions in area and power, respectively. The experimental results also illustrate the efficacy of using CA as a reservoir. Mapping the input signal into a higher-dimensional feature space, the rich dynamics recreated by CA can effectively improve the performance of the ensemble Bloom filter classifier. This model can completely eliminate expensive computational operations, such as floating-point calculation and integer multiplication. Owing to the simplicity of *EnsembleBloomCA*, our model shows promising potential for hardware implementation.

Chapter 3 contains material from "BloomCA: A Memory Efficient Reservoir Computing Hardware Implementation Using Cellular Automata and Ensemble Bloom Fil-

ter", by Dehua Liang, Masanori Hashimoto, and Hiromitsu Awano, which appears in Design, Automation & Test in Europe Conference & Exhibition (DATE), February 2021 [90]. The dissertation author was the primary investigator and author of this paper.

Chapter 3 contains material from "A Hardware Efficient Reservoir Computing System Using Cellular Automata and Ensemble Bloom Filter", by Dehua Liang, Jun Shiomi, Noriyuki Miura, Masanori Hashimoto, and Hiromitsu Awano, which appears in IEICE Transactions on Information and Systems, July 2022 [91]. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Binary HDC System utilizing Window Striding

4.1 Introduction

The emergence of the Internet of Things (IoT) has led to a copious amount of small connected edge devices and systems [9]. Many of these devices need to perform classification tasks such as speech recognition [69], activity recognition [70], and image classification [64]. Though Deep neural networks (DNNs) have provided high accuracy for complex classification tasks, with the scale of DNNs increasing, the high computational complexity and memory requirement of DNNs hinder usability to a broad variety of embedded applications. The energy constraint of edge devices hinders them from the real-time training of NN models [92].

Although sending the data to a powerful cloud platform to perform tasks is one of the options, there are still transmission delays and privacy security issues. For example, in health care monitoring, we often require learning algorithms to have real-time control of the patient's daily behavior, speech, and bio-medical sensors. Sending all data points to the cloud, cannot guarantee scalability and real-time response, which is often undesirable due to privacy and security concerns [24]. Hence, for edge devices with limited hardware resources, the demand for a more processing-efficient model is rising.

Brain-inspired hyperdimensional computing (HDC) has been proposed as a computing method that processes cognitive tasks in a more lightweight way [93]. HDC aims at realizing real-time performance and robustness through using strategies that more closely model the human brain [94]. HDC relies on mathematical properties of high-dimensional vector spaces and use high-dimensional distributed representations called hypervectors [95]. HDC works based on the existence of orthogonal hypervectors which can be combined using well-defined vector space operations. The mathematics governing the high dimensional space enables HDC to be easily applied to different learning

problems. The first step in HDC is to encode/map data points from the original domain to the high-dimensional space with bit-wise operations. During the training, HDC combines the encoded hypervectors to generate a hypervector representing each class. The classification task at inference performs by searching the similarity of an encoded test hypervector with all trained classes.

However, there are still several remaining challenges in the application of HDC in edge devices: (i) Huge memory cost for the encoding procedure. In most HDC models, they need to represent both the index and value of the input features via hypervectors, which requires a large size of memory blocks. Such a process takes huge occupation of the total inference memory cost, *e.g.*, 96.15% for letters recognition task. (ii) Many HDC algorithms need to be trained on expensive floating-point (FP) hypervectors and perform the inference with costly cosine similarity measurement, which leads to the increase of hardware resource requirement in edge devices. (iii) The hypervector of most HDC is generated holistically. For the hypervector with thousands of dimensions, optimization based on dimension-wise sparsity has been proposed in some prior works. However, feature-wise sparsity should also be under consideration when it comes to image processing tasks. (iv) The widely used retraining procedure in current HDC algorithms requires tens of iterations to get saturated, which leads to long training time.

In this work, we propose a novel HDC system: *StrideHD*. The uniqueness of *StrideHD* is to capture the critical features utilizing window striding method and organize the HDC architecture in a distributed way. After utilizing the window striding to chop the input images, thermometer binarization and max-pooling are applied. The extracted binary features are further encoded to hypervectors with high orthogonality, which enables efficient training/testing in our HDC model. Meanwhile, the hypervectors can be generated without the expensive item memory requirement. The main contributions of our *StrideHD* model are as follows:

- Successfully eliminated the costly Channel item Memory (CiM) and item Memory (iM) by exploiting a pseudo-random hypervector generation mechanism in the encoder. Besides the traditional dimension-wise sparsification, a feature-wise model sparsification is further proposed for image processing tasks.
- Compared to two HDC baselines, our single-pass training achieved a 27.6× and 8.2× reduction in memory cost without hurting the accuracy, while the iterative training can further improve 8.7× memory efficiency.
- Under the same inference memory cost, the classification accuracy of single-pass mode *StrideHD* is averagely 13.56% higher than the baseline HDC models.
- As an extension, we propose an iterative retraining mode in our *StrideHD*, which averagely provides 11.33% accuracy improvement to its single-pass mode in *DistriHD* [96], which can be accomplished in fewer iterations compared to the other baseline HD models.

- For hardware cost, we achieved over 9.9× and 28.8× reduction compared with baseline HD [24] in area and power, respectively.

4.2 Proposed Method

As a novel HDC system, *StrideHD* utilizes window striding to capture the critical features in the distributed way, and enables to train/test the model with such distributed binary hypervectors. Fig. 4.1(a) shows the overview of *StrideHD* for image classification task. In *StrideHD*, the first step is to extract the feature from the original images, in which we apply the window striding technique. Utilizing the receptive window striding, the critical locality patterns can be captured effectively. Then, for features in each receptive window, we apply maxpooling layer and thermometer encoding method to quantize the non-binary values into binary data [90]. After obtaining the binary features for each receptive window, we apply the hypervector encoder to convert features into hypervectors. Note here that, contrary to the conventional HD, a single input image is converted into multiple (*i.e.*, *distributed*) hypervectors. During the initial single-pass training, these distributed hypervectors are combined in a training module in order to create a set of binary hypervector representing each class. The classification is performed by finding the class distributed hypervectors set which has the highest similarity with the test distributed hypervectors set. Note again that the similarities are respectively computed for each hypervectors. Further, for the model size reduction, we prune part of the class distributed hypervector by using validation dataset whose procedure is detailed in Sec. 4.2.5. Since *StrideHD* works with a binary model, the inference can be performed with hardware-friendly Hamming distance as the similarity matrix. As an extension, we also proposed iterative learning in our framework for performance improvement. The pseudo-code for the *StrideHD* is further shown as the Algorithm 1 and the Algorithm 2. In the following, we explain the details of the *StrideHD* functionality.

4.2.1 Feature Extraction by Window Striding

For the input image, we firstly apply a $W_{rec} \times H_{rec}$ size receptive field with a stride of T_{rec}, *i.e.*, chopping the pixels with striding windows. Then we apply a max-pooling operation to improve the generalization of the model. After these two steps, L_r distributed patterns with non-binary elements are generated. Subsequently, we binarize the non-binary element with thermometer way [90]. Assume the range of non-binary element u is $\Delta u = u_{max} - u_{min}$ and quantize it into L_b levels as follow:

$$u^{(b)} = \begin{cases} 1, \frac{\Delta u \cdot b}{L_b} < u, \\ 0, \frac{\Delta u \cdot b}{L_b} \geq u. \end{cases} \tag{4.1}$$

Algorithm 1: *StrideHD* Computing framework

Design Parameters: shape of striding window $W_{rec} \times H_{rec}$, number of training subsets L_e, number of dimension-wise selected distributed hypervectors M_s, number of feature-wise selected distributed hypervectors L_s, thermometer binarized levels L_b, number of dimensions D.

Require: Initialize an integer model $\mathbb{C}$ and a binary model $\mathbb{B}$ in shape of $N \times M \times L \times D$ with '0's, where N is the number of categories, M is the number of dimension-wise distributed hypervectors, L is the number of feature-wise distributed hypervectors.

Ensure: Update the binary model $\mathbb{B}=sgn(\mathbb{C})$

{① Single-pass & Iterative Training}

Split training set to L_e minibatch including samples u^e with $N_{feature}$ non-binary features $u(n)$, which corresponds to the labels $y(u)$.

$u^{(e)}(n) \leftarrow u(n), e=1,2,...,L_e$

for $e=1$ to L_e **do**

  for $n=1$ to $N_{feature}$ **do**

    # *thermometer binarization*

    $u^{(e,b)}(n) \leftarrow u^{(e)}(n), b=1,2,...,L_b$ using Eq.(4.1)

    for $b=1$ to L_b **do**

      # *window striding & max-pooling*

      $u^{(e,b,l)}(n') \leftarrow u^{(e,b)}(n), l=1,...,L_b; n'=1,...,W_{rec} \times H_{rec}$

Reshape $u^{(e,b,l)}(n')$ as $f_n^l, l \in [0,L], n \in [1, W_{rec} \times H_{rec}]$

where $L=L_e \times L_b \times L_r$

Generate a random matrix $R_{(i,j)} \in [1, W_{rec} \times H_{rec}]^{M \times B}$

for $i=1$ to M **do**

  $p_i \leftarrow \sum_{j=1}^{B} f_{R_{(i,j)}} 2^{B-j}$ using Eq.(4.3)

  for $l=1$ to L **do**

    if *single-pass training* **then**

      for $k=1$ to N **do** # *all categories*

        $\mathbb{C}_k^l[p_i] \leftarrow \mathbb{C}_k^l[p_i]+1$

    else if *iterative training* **then**

      if $y_{predict} \neq y$ **then**

        $\mathbb{C}_{correct}^l[p_i] \leftarrow \mathbb{C}_{correct}^l[p_i]+1$ # *correct category*

        $\mathbb{C}_{predict}^l[p_i] \leftarrow \mathbb{C}_{predict}^l[p_i]-1$ # *predict category*

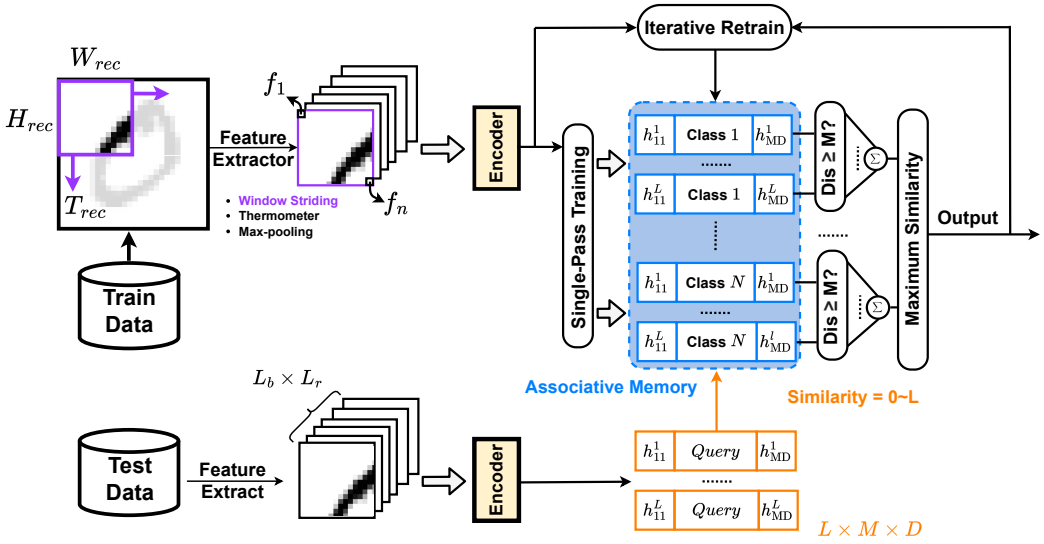

Figure 4.1: Overview of the proposed *StrideHD*.

where $b \in [1, L_b]$. In this way, $L_r \times L_b$ distributed binary patterns are extracted from each image.

4.2.2 Encoder

Considering the expensive memory cost of item Memory, we proposed an encoder with a pseudo-random hypervector generation mechanism that can map a pattern to a hypervector only using the shifting operation. Fig. 4.2 shows the overview of the proposed encoder, and Fig. 4.3 shows the orthogonality comparison between the hypervectors generated by different encoders. Assuming the l-th binary pattern ($l \in [1, L_b \times L_r]$) obtained via feature extraction is represented by vector F^l. Where $F^l = \{f_1, f_1, ..., f_n\}$ with n elements ($f_i \in \mathbb{N}$), will be mapped to the distributed hypervector $H^l = \{h_1^l, h_2^l, ..., h_D^l\}$ with D dimensions ($h_i \in \{0,1\}^D$). Firstly, we randomly select the elements from vector F^l and construct a matrix R:

$$R = \begin{pmatrix} f_{\sigma(1,1)} & f_{\sigma(1,2)} & \cdots & f_{\sigma(1,B)} \\ f_{\sigma(2,1)} & f_{\sigma(2,2)} & \cdots & f_{\sigma(2,B)} \\ \vdots & \vdots & \ddots & \vdots \\ f_{\sigma(M,1)} & f_{\sigma(M,2)} & \cdots & f_{\sigma(M,B)} \end{pmatrix} \tag{4.2}$$

where $f_{\sigma_{(i,j)}} \in \{f_1, f_1, ..., f_n\}$, $i \in [1, M]$, $j \in [1, B]$. M and B represent the number and range of the generated binary numbers, respectively. And the function σ represents the

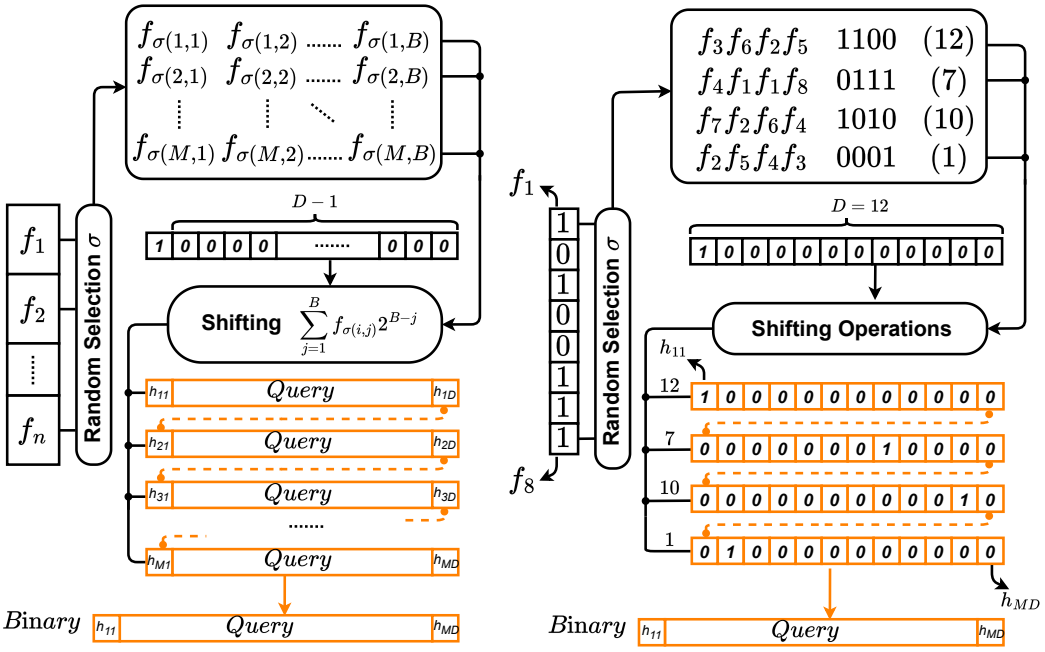

Figure 4.2: Overview of the proposed encoder in *StrideHD*.

random selection. Each row of the matrix R can be considered as a binary number p_i:

$$p_i = \sum_{j=1}^{B} f_{\sigma_{(i,j)}} 2^{B-j} \tag{4.3}$$

Similar to Bloom filter [90], we can generate M binary vectors in the range of B-bits by flipping the p_i-th bit of the empty vector (all logic "0"), which can be easily accomplished with shifting operation. Finally, the distributed hypervector H^l is the connection of these binary vectors.

Such a mechanism has three main advantages as compared to the other encoding methods in the HDC algorithms [24, 37–42]. First, unlike existing approaches which need to read the index hypervectors from item Memory, this encoding method can generate hypervectors only using shifting operation. The expensive memory cost by item Memory (shown in table 2.2) can be avoided. Second, this method doesn't need to do accumulation and majority voting for each dimension during encoding, which means it is hardware-friendly and suitable for parallel implementation. Third, the hypervectors generation with high orthogonality. We randomly select 1,000 training images from MNIST dataset and generate the hypervectors with different encoders. As shown in Fig. 4.3, the orthogonality measure is based on Hamming distance, while our encoder achieved superior performance.

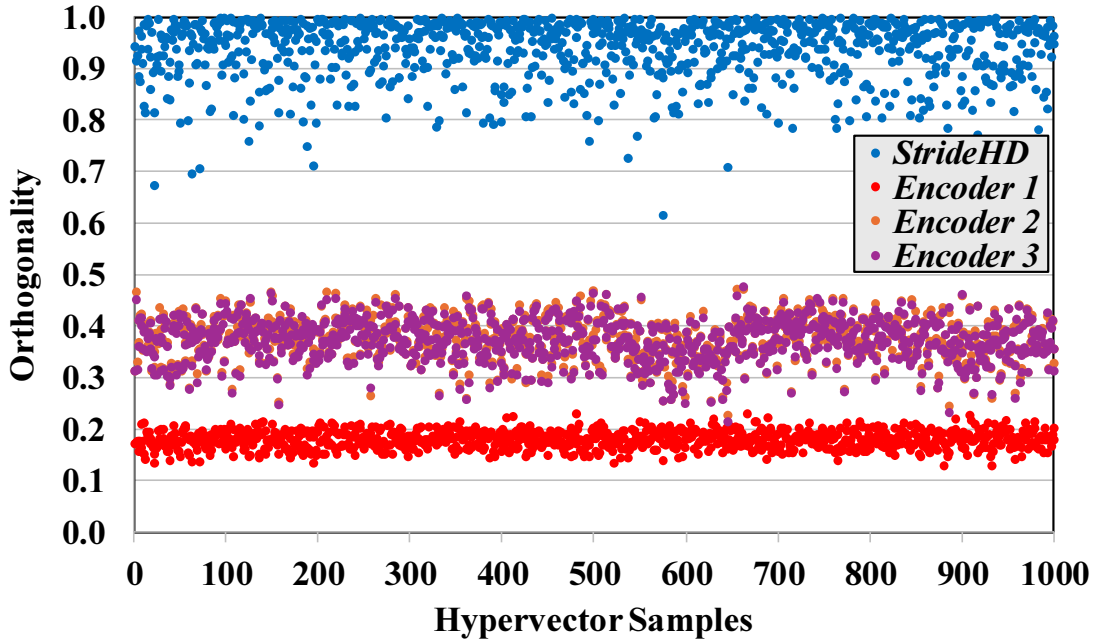

Figure 4.3: Hypervector Orthogonality of different Encoders.

4.2.3 Single-Pass Training

Initially the AM is blank, *i.e.*, all the values are set to logic "0". After the feature extraction, $L_r \times L_b$ distributed binary patterns are obtained. By using the record-based encoder we described in Chapter 2, a set of distributed hypervectors H^l are generated ($l \in [1, L_b \times L_r]$). For all input within the same class, the training data will be divided into L_e training subset. For all the data within the same training subset $\{S_1, ..., S_K\}$, the distributed hypervectors are added to create the class distributed hypervectors $\mathbb{C} = \{\mathbb{C}^1, \mathbb{C}^2, ..., \mathbb{C}^L\}$ with integer addition for each dimension as follow:

$$\mathbb{C}^l = H_1^l + H_2^l + ... + H_K^l \tag{4.4}$$

where $\mathbb{C}_i^l$ represents the l-th distributed hypervector belonging to the i-th class and $L = L_b \times L_r \times L_e$. These class distributed hypervectors will be stored in the AM during the training. Note that in comparison with the existing HD model, the majority voting is simplified to OR operation, which contributes to a fast and ultra-efficient learning process.

4.2.4 Inference

After the feature extraction and encoding procedure, L distributed hypervectors H^l are generated to represent one input image for query. Similarly, L distributed class hypervectors $\mathbb{C}^l$ have been trained and stored in the AM part, which contains the universal information of each class in HD space. When it comes to the inference, we measure the similarity of them as follows:

$$Similarity = \sum_{l=1}^{L} sgn[\delta(H^l \& \mathbb{C}^l, \mathbb{C}^l) - M]. \tag{4.5}$$

Where & represents the bitwise AND operation and the δ is the Hamming distance between two vectors. The *sgn* represents the sign function that extracts the sign of a real number. The class of distributed hypervectors with the highest similarity is chosen as a representative category of the testing image.

4.2.5 Optimized Model Sparsification

To improve memory efficiency, we proposed to sparsify associative memory with the labeled validation data. Compared with the existing mechanism, our model sparsification is changed from unsupervised to supervised mode, which are performed dimension-wise and feature-wise. The details are shown in Fig. 4.4.

For dimension-wise sparsity, we use the labeled validation images to calculate the accuracy of each dimension. In the ideal scenario, the value of a dimension in the validation hypervector should be equal to the corresponding dimension in the matching class hypervector, while all other values of this dimension in the mismatching class hypervectors should not be equal to it. After obtaining the accuracy performance of dimensions, the dimensions with the lowest accuracy are selected to be dropped from the HDC model as they have the least/worst impact on differentiating the classes. Note that in our proposed encoding, the distributed hypervector H^l is a connection of the shifted binary vectors, which means the dimensions in these shifted binary vectors can not be dropped separately. Hence, the discard of dimensions only happened for the whole shifted binary vectors, which makes the dimensions of class distributed hypervectors decrease: $\mathbb{C}^l \in \{1,0\}^{M \cdot D} \rightarrow \mathbb{C}^l \in \{1,0\}^{M_s \cdot D}$.

Similarly, for the feature-wise sparsity, we can also calculate the accuracy for class distributed hypervector by validation images simultaneously. The $\mathbb{C}^l$ with the lowest accuracy are selected to be dropped as those represented features have the least impact on differentiating the classes. The number of $\mathbb{C}^l$ is decreased: $l \in [0, L] \rightarrow l \in [0, L_s]$.

Algorithm 2: Sparsification & Inference

{② Sparsification}

Calculate the accuracy of distributed hypervectors $\mathbb{C}^l[p_i]$ where $i \in [0,M], l \in [0,L]$, discard the $\mathbb{C}^l[p_i]$ with poor Acc. Model $\mathbb{C}$ and $\mathbb{B}$ are compressed as:

$M \rightarrow M_s, L \rightarrow L_s$

{③ Inference/Prediction}

for k=1 to N **do**

  for l=1 to L_s **do** # *before* ②:L; *after* ②:L_s

    $Similarity(k)$ += AND($\mathbb{C}_k^l[p_i]$), i=1,2,...,M_s

    # *before* ②:M; *after* ②:M_s

Output← $Argmax(Similarity)$

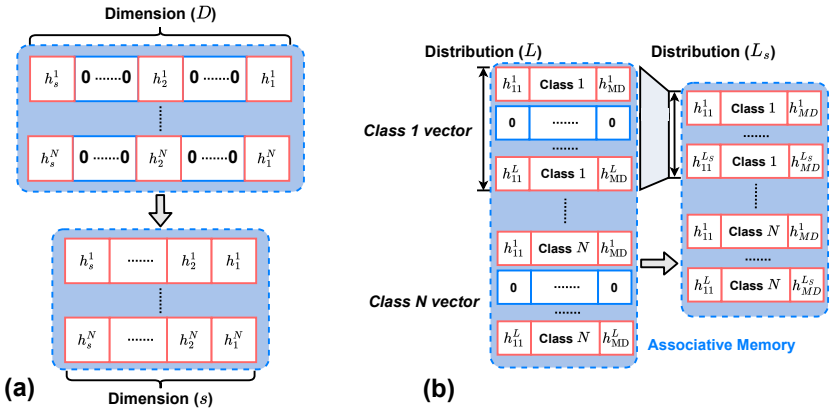

Figure 4.4: (a) Dimension-wise Sparsification. (b) Feature-wise Sparsification.

4.2.6 Iterative Retraining

As an extension, we also proposed iterative learning in our framework, which aims at reducing the error rate of the initial HD model by employing gradient descent. As shown in Figure. 4.1, the *StrideHD* firstly encodes the training data to query distributed hypervectors, then check its similarity with each pre-stored class distributed hypervector set as mentioned in Subsection 4.2.4. If the class distributed hypervector set with the highest similarity matches the correct label, the *StrideHD* ignores updating the model. However, if an encoded training data H incorrectly matches with the model, we add this query to the correct class $\mathbb{C}_{correct}$ while subtracting it from the predicted incorrect class $\mathbb{C}_{predict}$ as follows:

$$\begin{cases} \mathbb{C}_{correct} = \mathbb{C}_{correct}[+]H \\ \mathbb{C}_{predict} = \mathbb{C}_{predict}[-]H \end{cases} \tag{4.6}$$

Where $[+]$ and $[-]$ is a binary addition and subtraction for each dimension. Note that such accumulation is performed in class distributed hypervectors with integer elements in a pre-defined range, which is also considered as the counters model [24]. When it comes to the inference, we can use an additional binarized model with the same size of dimensions. All dimensions with a smaller value than 0 are assigned to 0, while other elements are assigned to 1.

Based on the novel encoding procedure described in 4.2.2, the percentage of logic '1' in the encoded hypervector of *StrideHD* is much less than the conventional hypervector. Hence, the update of model mainly focuses on the more significant dimensions, which leads to efficient iterative retraining and fewer iterations.

4.3 Hardware Implementation of Inference

The hardware implementation of *StrideHD* during inference mainly consist of four different blocks: (❶) Encoding, (❷) Associative Memory Blocks, (❺) AND Gates Array, and (❻) Nearest Distance Searching modules. Figure 4.5 shows the overview of our hardware architecture.

4.3.1 Encoding Blocks

The hardware implementation of each encoding block is shown in Figure 4.5 (❹). This process performs the (❸) feature extraction and pseudo-random hypervectors generation mechanism with the memory address decoder, which are integrated in the (❶) as hardware implementation of the Encoding Blocks. Mathematically, the computation of feature extraction can be performed by the window striding, thermometer binarization, and max-pooling techniques.

- As for the window striding, it is mainly implemented with the wire connection, which does not require additional transistors in hardware implementation.
- The next step is to convert each original decimal feature to the L_b-bits thermometer binary data. This process can be accomplished with cheap combinational logic circuits. According to Eq.(4.1), the more continuous '1's at the beginning of the converted data represent the larger value of the original feature.
- Based on the characteristic of thermometer binary data, it is convenient for the implementation of the max-pooling layer, which can be performed by bit-wise OR gates for each bit of the binary data.

Figure 4.5(❸) shows an example of thermometer binarization and its corresponding max-pooling operation. After this step, each original data point is converted to L_s different M_s-bits binary patterns, which are the input of the address decoders. Based on the decoded addresses, the data from the corresponding memory cells in C different

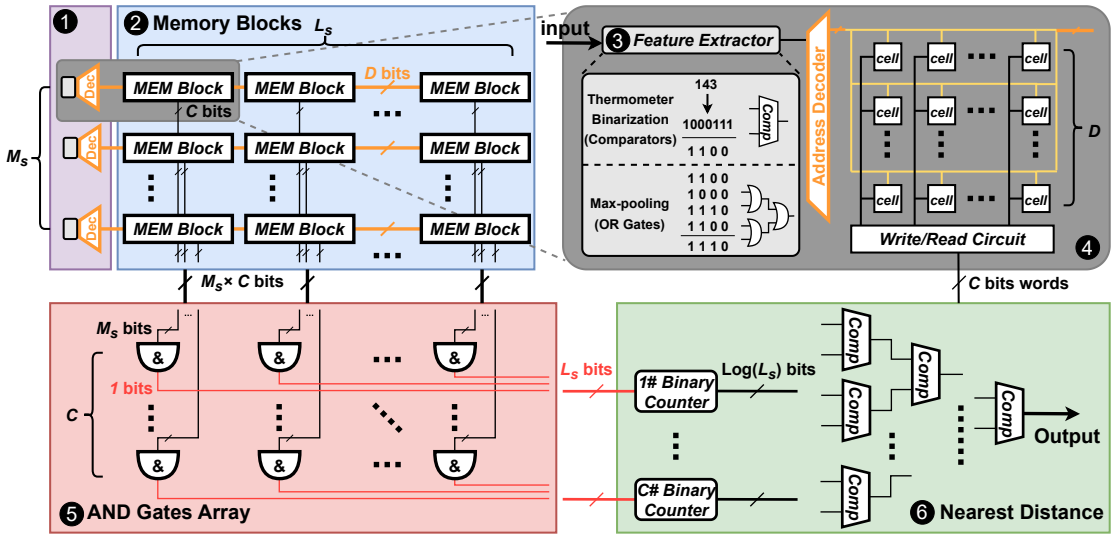

Figure 4.5: The hardware implementation of *StrideHD* during inference includes Encoding, Associative Memory Blocks, AND Gates Array, and Nearest Distance Searching modules.

categories are read. Note that the L_s and M_s represents the number of feature-wise and dimension-wise distributed hypervectors after the model sparsification, which has been accomplished by validation data during the training. Such model sparsification process aims at further reducing the memory usage of well-trained model, leading to a light weight requirement for hardware implementation during the inference.

4.3.2 Associative Memory (AM) Blocks

Figure 4.5(❷) shows the implementation of the AM Blocks. Unlike the prior HDC algorithms that read the data from all the memory cells, the *StrideHD* only requires reading parts of the memory cells. For each inference operation, $L_s \times M_s \times C$ bits of data are read from the AM Blocks, while the AM Blocks contain D times larger memory in total. Based on the decoded addresses, the data from the corresponding memory cells are read, which leads to M_s different C-bits data as the input of the following AND gates array. Note that the readout data are not the hypervector of each category but represent the similarity measure results.

4.3.3 AND Gates Array

Figure 4.5(❺) shows the implementation of the AND gates array. Compared to the XOR gates array in prior works, even costing the same number of memory cells in AM Blocks, the scale of AND gates array in *StrideHD* is much smaller. On one hand,

the scale of readout data from AM is reduced for D times. On the other hand, the *StrideHD* are checking whether the readout dimensions are '+1', while the traditional way (Encoder I, II, and III) requires to check whether the readout dimensions are equal to query hypervector. Hence, readout data from AM blocks are the only input of AND gates array, while the two inputs of the traditional XOR gates array are the readout data and query hypervector, respectively. When M_s-bits of readout data from the same category are '+1', the similarity of the corresponding category is increased by 1, which makes the similarity in the range from 0 to L_s.

4.3.4 Nearest Distance Searching

Figure 4.5(❻) shows the implementation of the nearest distance searching module. After getting $C \times L_s$ bits data from AND gates array, it requires C different binary counter to calculate the number of '+1' in the L_s-bits data, which is considered the similarity for each category. Finally, utilizing the comparators to get the output. The range of similarity in *StrideHD* is much smaller than the traditional one ($D >> L_s$).

4.4 Experiment

4.4.1 Experimental Setup

We consider the popular HDC algorithm [24] as the baseline, which is similarly utilizing binary hypervectors and eliminating FP calculations. We evaluated *StrideHD* and baseline HD training and inference with three encoders on an Intel Core i7 7600 CPU using an optimized C++ implementation. To verify recognition quality of *StrideHD*, we consider three problems: MNIST [64], Kuzushiji-MNIST [97], Fashion-MNIST [98], and SMILES [71]. For the MNIST-kind datasets, we randomly selected 55k images for training and 5k images for validation.

MNIST: is a collection of handwritten digits in grayscale format and is intensively used to compare the performance of many classification models.

Kuzushiji-MNIST: is a dataset that focuses on Kuzushiji (cursive Japanese) [97]. Even

Table 4.1: Parameters Setting for *StrideHD*.

	W_{rec}	T_{rec}	L_b	L_e	D	S_D	S_F
***mnist**	5	3	4	2	150	95%	20%
smiles	7	5	4	2	150	95%	20%

though this dataset is created as a drop-in replacement for the MNIST dataset, the characteristics of Kuzushiji and Arabic numbers are completely different, which makes it more challenging than MNIST.

Fashion-MNIST: is a new dataset comprising of fashion products, such as shirts, T-shirts, or coats that look very similar at 28×28 pixel resolution in grayscale, making many samples ambiguous even for humans (Human performance on Fashion-MNIST is only 83.5% [98]).

SMILES: is a face recognition task that aims at classifying the images with or without smiling. There are 13,165 images in the dataset, with each image having a size of 64×64 pixels. Among all the face images, 9475 of these examples are not smiling, while only 3,690 belong to the smiling class. Hence, we randomly select 600 positive images 600 negative images for testing, and 300 images for the validation.

In software, we utilize the MNIST dataset to explore the impact of several design parameters in *StrideHD* single-pass training, and the reduction of the inference memory cost are evaluated in comparison with the baseline HD. As an extension, we make a comparison between our proposed *StrideHD*, and three baseline HD algorithms. The BinHD, SecureHD, and DUAL represent the iterative HDC learning framework in [24] utilizing the Record-based Encoder I from [24], Random-projection Encoder II from [37], and Non-linear Encoder III from [68], respectively.

4.4.2 Single-Pass Training Parameters Tuning

In our experiment for the MNIST dataset, the max-pooling layer is selected to have a stride of 2, a squared window of size 2, and zero padding. According to the evaluation results shown in Fig. 4.6(a), we applied a 5×5 receptive window with a stride of 3. Meanwhile, Fig. 4.6(b-c) illustrates the impact of binary levels L_b and the number of training subsets L_e. With higher L_b and L_e, the classification accuracy is improved with a larger memory cost. Hence, for competitive accuracy and memory efficiency, we choose $L_b = 4$ and $L_e = 2$ as a performance trade-off. Fig. 4.6(d-f) shows the classification accuracy under different Feature-wise sparsity, Dimension-wise sparsity, and the dimensions D. Similarly, we adopt the Feature-wise and Dimension-wise sparsity as 20% and 95%, respectively. The parameters setting of our *StrideHD* in different tasks are listed in Table 5.1.

4.4.3 Memory Efficiency

As we mentioned in Table 2.2, the iM and CiM are averagely taking huge occupation of memory cost (93.4% and 4.7%) during inference for the BinHD, which is unnecessary in the *StrideHD*. Based on the observation of Table 2.1 and Table 2.2, we found that the hardware cost for Encoder III in DUAL is much higher than the Encoder I in BinHD and

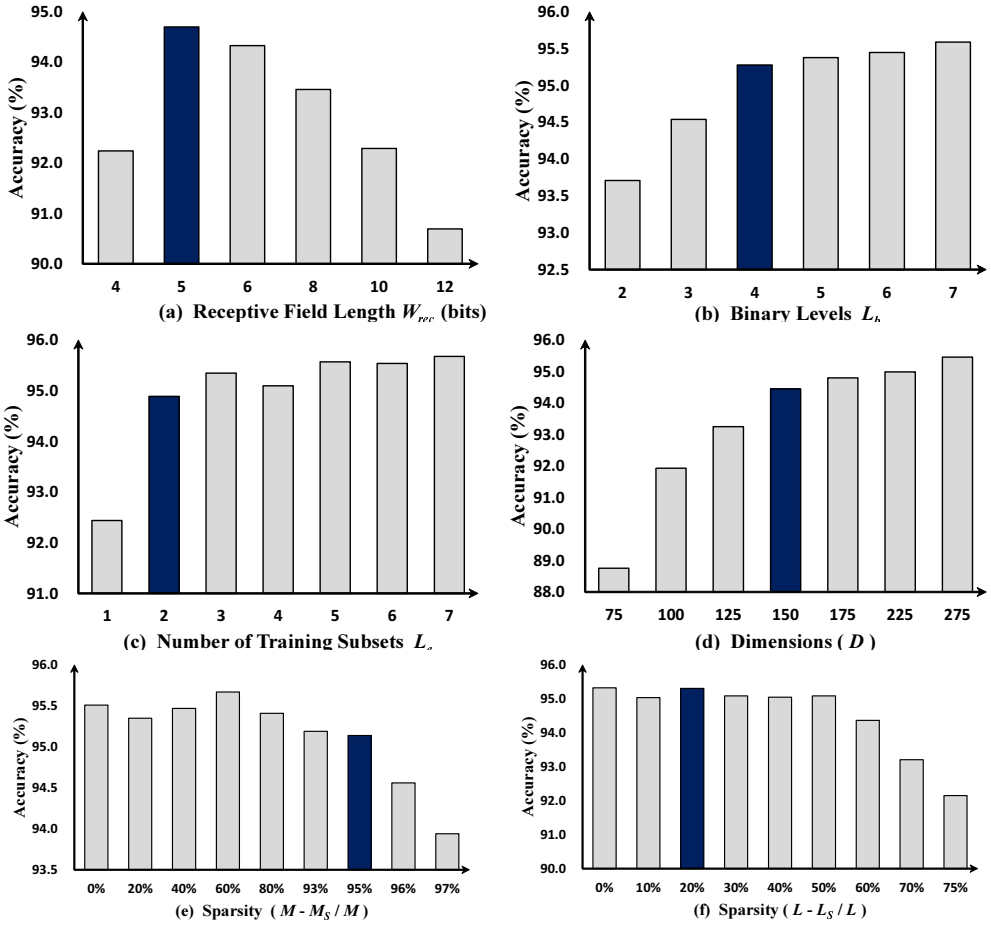

Figure 4.6: Impact of Different Parameters: (a) Length of Receptive Window. (b) Binary Levels L_b. (c) Number of Training Subsets L_e. (d) Dimensions D. (e) Dimension-wise Sparsity. (f) Feature-wise Sparsity.

Encoder II in SecureHD due to the expensive FP calculation and high data precision. Hence, we make a comparison of *StrideHD*, BinHD, and SecureHD in terms of classification accuracy and memory cost, which is shown in Fig. 4.7. For a comprehensive and fair comparison, we evaluated the performance of the baselines in two different ways.

The first way is to calculate the inference memory cost including the CiM and iM part, which performs very poor accuracy in Fig. 4.7 (a). To achieve the same level of accuracy (*e.g.*, 94.8%), the iterative BinHD and SecureHD require 27.6× and 8.2× memory cost compared to the single-pass training *StrideHD* model. When applying iterative learning to our proposed model, the accuracy is improved compared to the single-pass training, which results in 8.7× memory efficiency. The significant memory reduction mainly comes from the elimination of the costly CiM and iM.

The second way is to calculate the baseline memory cost without the expensive CiM

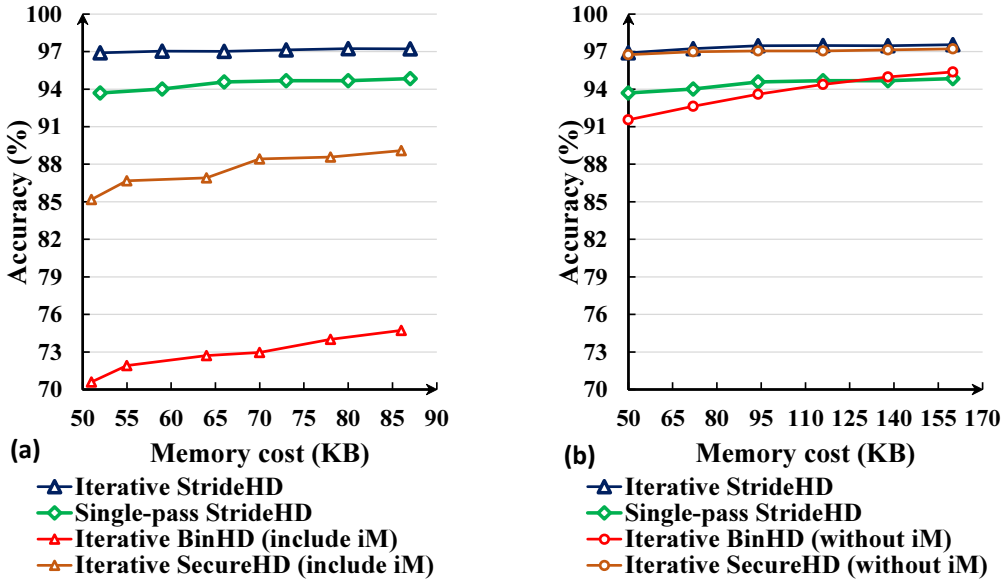

Figure 4.7: The comparison of accuracy and memory cost.

and iM parts, which can give a fair comparison to the performance of the trained classifiers. Although increasing the number of dimension D in BinHD and SecureHD results in improving the classification accuracy, but also leads to a huge usage of memory during inference. In MNIST dataset, when D is increased to 10K, the accuracy of the iterative baseline HDC models gets saturated at around 96%, which consumes 97.7 KB for the associative memory (AM) part and 7.7 MB for the total memory cost. We found that with the same usage of AM, our single-pass training has the advantage over the iterative BinHD algorithm but is not competitive with the iterative SecureHD. When we apply iterative learning to our proposed model, the accuracy of *StrideHD* is improved to the same level as SecureHD. Hence, such experimental results show that our method successfully eliminated the expensive CiM and iM while maintaining the same accuracy as the classifier.

4.4.4 Retraining Iterations & Classification Accuracy

Besides the huge memory efficiency, the fast training process is also considered as the advantage of *StrideHD*. Fig. 5.5 shows the comparison during the iterative training. The parameters setting of *StrideHD* is shown in Table 5.1, while the $D = 10k$ for the baseline HDC models. Compared to the baseline HD algorithms, our method requires much fewer iterations to achieve saturated and stable classification accuracy. Since the percentage of logic '1' in the encoded hypervector of *StrideHD* is much less than the conventional hypervecotor in baseline HD algorithms [24, 37, 68]. Therefore, the update of the model can mainly focus on the more significant dimensions, which leads to

Table 4.2: Classification Accuracy Comparison between *StrideHD* and baseline HDC models for different datasets.

		MNIST	K-MNIST	F-MNIST	SMILES	Average
Single-Pass	***BinHD***	80.48%	49.41%	70.54%	69.83%	67.57%
	SecureHD	81.42%	51.01%	67.06%	75.33%	68.71%
	DUAL	81.25%	51.16%	66.87%	75.17%	68.61%
	StrideHD	**94.52%**	**75.34%**	**77.65%**	**79.92%**	**81.86%**
	Improvement	**13.47%**	**24.81%**	**9.49%**	**6.48%**	**13.56%**
Iterative	***BinHD***	93.57%	69.16%	82.62%	88.17%	83.38%
	SecureHD	94.87%	78.31%	83.06%	86.09%	85.58%
	DUAL	93.01%	79.76%	81.32%	86.17%	85.07%
	StrideHD	**97.51%**	**90.14%**	**86.67%**	**98.42%**	**93.19%**
	Improvement	**3.69%**	**14.40%**	**4.34%**	**11.61%**	**8.51%**

efficient iterative retraining and fewer required iterations. Meanwhile, most traditional HDC algorithms simply calculating the Hamming distance during the inference query. Although the iterative learning mode of HDC tends to increase the similarity between the class hypervectors and training hypervectors, some of the dimensions still changes back and forth. The noisy information within each training hypervector are accumulated within such dimensions, which might result in the performance fluctuations of the HDC model. For our *StrideHD*, the bit-wise AND operation and sign function are included during hypervector query, which provides a threshold to reduce the noisy information during the retraining. For the single-pass training mode, the classification accuracy of *StrideHD* is averagely 13.56% higher than the baseline HDC models. As an extension, the iterative retraining procedure averagely provides an 11.33% accuracy improvement to the single-pass *StrideHD* model. The comparison of classification accuracy is shown in Table 4.2.

Overall, in comparison with the popular HDC models BinHD, SecureHD, and DUAL, our *StrideHD* significantly reduced the memory cost for the inference by the elimination of costly item Memory. Meanwhile, the fast and simplicity of the training process avoids expensive iterative training while maintaining the same level of classification accuracy.

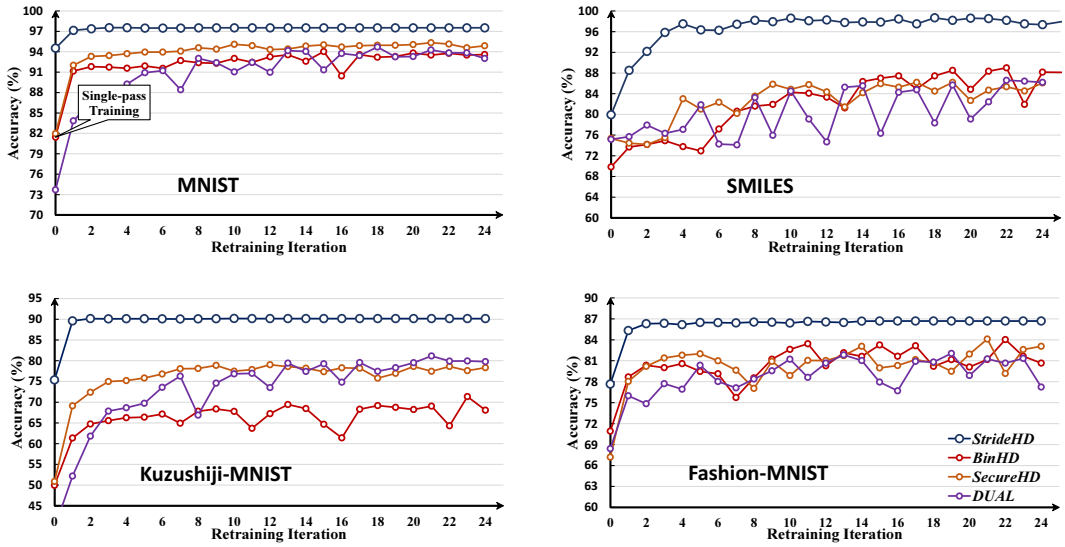

Figure 4.8: Comparison of Iterative Retraining for *StrideHD* and Baseline HD models in different datasets.

4.4.5 Hardware Implementation

The hardware architecture and functionality of *StrideHD* and BinHD are designed via RTL SystemVerilog. Then we use *Synopsys Design Compiler* to synthesize and report the area and power consumption in 65-nm ASIC flow. All the synthesis are based on minimum hardware area cost approach, and the clock period are set as 5 nanoseconds. The memory part can be individually simulated with *CACTI* [99]. The memory type is also chosen to be the main memory in 65-nm ASIC flow, which doesn't contain any tag array and every access will happen at page granularity.

Table 4.3 shows the comparison of *StrideHD* and the BinHD during inference in terms of ASIC area and power consumption with the same level of classification accuracy (94.8%). The maximum propagation delay of *StideHD* and BinHD is 4.61 and 4.83 nanoseconds, respectively. There are no timing violations. Both *StrideHD* and baseline are using the binary mode, which mostly exploits the hardware-friendly Hamming distance for similarity measurement.

The memory block takes over 96.9% of the area and 99.1% of the power consumption in the *StrideHD* model. The gap in hardware cost between the proposed model and the baseline mainly comes from memory efficiency. The key concept behind *StrideHD* is to eliminate the costly memory blocks in HDC, *i.e.*, the CiM and the iM blocks, which provides significant energy consumption reduction. We aim for this HDC model to bring benefits to customers, irrespective of the type of memory used in hardware implementation. Hence, to exclude the performance gap of the memory, we didn't manually design the memory part for *StrideHD* and synthesize it alone with the logic parts, but simulate

Table 4.3: Hardware Performance Comparison.

	Baseline	***StrideHD***	**Reduction**
Area (mm^2)	31.88	3.22	9.9×
Power (mW)	3839	133.2	28.8×

it by the architectural simulation model *CACTI* individually. Both of the baseline HD architecture and our *StrideHD* are compared using the same method to evaluate the power/area performance of the memory block, the accuracy of *CACTI* does not affect the comparison of the hardware cost. Another reason is that the baseline requires the majority voting mechanism to generate the hypervectors while the encoding in *StrideHD* only requires shifting and OR operations, which also contributes to the improvement of hardware efficiency.

4.5 Conclusion

In this work, we proposed a novel HDC system *StrideHD* that utilizes window striding to capture the locality feature of images. This framework supports using binary hypervectors and achieves high accuracy with fast training speed and significantly low hardware cost. Compared to the baseline BinHD and SecureHD utilizing iterative learning strategy, our framework achieves a 27.6× and 8.2× reduction in memory cost without hurting the accuracy in single-pass mode, while the iterative training can further provide 8.7× memory efficiency. Under the same inference memory cost, the accuracy of single-pass mode *StrideHD* is averagely 13.56% higher than the baseline HDC. As an extension, the iterative retraining mode of *StrideHD* averagely provides 11.33% accuracy improvement to its single-pass mode, which can be accomplished in fewer iterations compared to the baseline HDC. Our hardware evaluation results demonstrate that compared to BinHD, our *StrideHD* achieves over 9.9× and 28.8× reduction in area and power, respectively.

The experiment result illustrates that the improvement of memory efficiency mainly comes from the innovation of hypervecter generation. Our *StrideHD* architecture utilizes window striding to capture the critical features in a distributed way. By proposing a pseudo-random hypervector generator as the encoder in the HDC system, the input images can be converted to distributed hypervectors without the requirement of the costly CiM and iM while maintaining high orthogonality, which enables efficient training/testing in our HDC model. Since the Channel item Memory (CiM) and item Memory (iM) usually occupy over 95% of the memory requirement of the whole HDC model, the elimination of those parts leads to a significant improvement in memory efficiency.

The distribution of hypervectors based on features also constrains the noisy information from the less important pixels or binary channels. On the other hand, there is a trade-off between memory requirement and accuracy performance. Both our proposed method and the baseline HDC model can achieve higher classification accuracy when increasing the length of the hypervector, which also results in the expensive memory cost. Hence, the improvement of memory efficiency leads to better accuracy performance.

Chapter 4 contains material from "DistriHD: A Memory Efficient Distributed Binary Hyperdimensional Computing Architecture for Image Classification", by Dehua Liang, Jun Shiomi, Noriyuki Miura, and Hiromitsu Awano, which appears in Proceedings of the 27th Asia and South Pacific Design Automation Conference (ASP-DAC), January 2022 [96]. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "StrideHD: A Binary Hyperdimensional Computing System Utilizing Window Striding for Image Classification", by Dehua Liang, Jun Shiomi, Noriyuki Miura, and Hiromitsu Awano, which appears in IEEE Open Journal of Circuits and Systems, May 2024 [100]. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Robust HDC System for Voltage-scaled Circuits

5.1 Introduction

The emergence of the Internet of Things (IoT) has led to a copious amount of small connected edge-oriented devices and systems [9]. Many of these devices need to perform classification tasks such as speech recognition [69], activity recognition [39,70,74], and image classification [64, 73, 90, 91, 96]. However, most of these small edge-oriented systems do not have the sufficient computing power to accomplish the training process of sophisticated classification algorithms such as Deep Neural Networks (DNNs) individually. Even sending the data to a powerful cloud platform to perform tasks, there are still transmission delays and privacy security issues. For example, in health care monitoring, we often require learning algorithms to have real-time control of the patient's daily behavior, speech, and bio-medical sensors. Sending all data points to the cloud, cannot guarantee scalability and real-time response, which is often undesirable due to privacy and security concerns [24]. Hence, utilizing the cloud computing platform to run the learning algorithm, and then downloading these well-trained models back to edge-oriented systems is another efficient solution.

For practical implementation in real-life applications, energy consumption is one of the key issues in today's power-constrained edge-oriented systems. Ref. [47] points out that the energy consumption of LSI circuits can be minimized if the supply voltage is downscaled to the sub-threshold region. For example, the minimum energy consumption can be typically found below a 400 mV supply voltage for a microprocessor [101], or an FFT processor [102]. However, the aggressive voltage decrease also involves an increase in the circuit delay and raises the possibility of functional failure due to process variation [55, 56]. Meanwhile, today's machine learning platforms have major robustness issues dealing with insecure and unreliable memory systems. In traditional data

representation, when the circuit failure occurs in the memory systems, it may lead to the flipping of the exponent or most significant bits. Such an issue can increase the weight value to extremely large, thus changing the prediction result of machine learning models. Prior work [103] showed how a few bit flips on the DNNs model can result in a major change in the prediction result. Unfortunately, most existing learning solutions are sensitive to memory functional failures induced by possible noise, bit-flip attacks, or voltage scaling. To solve these new issues, the demand for a more lightweight algorithm with sufficient accuracy and ultra robustness is crucially raising.

To closer model the human brain, many researchers proposed HyperDimensional Computing (HDC) as an alternative computing method, which mimics important brain functionalities towards energy-efficient and noise-tolerant computing paradigm. HDC is motivated by the observation that the human brain operates on high-dimensional representations of data [104]. It performs computation on ultra-wide words, which can be considered as very high-dimensional vectors, or hypervectors. HDC works based on the existence of a huge number of hypervectors that can be combined using well-defined operations. The mathematics governing the high dimensional space enables HD to be easily applied to different learning problems. The first step in HD computing is to encode/map data points from the original domain to the high-dimensional space. During the training phase, HD combines the encoded hypervectors to generate a hypervector representing each class. The classification task at inference performs by checking the similarity of an encoded test hypervector with all trained classes. The HDC system exploits a redundant and holographic representation, ensuring all bits have the same impact on computing, which endues the model with potentiality against serious memory failure [39].

To overcome the performance degradation induced by voltage scaling while guaranteeing edge-oriented systems for long-term operations, a dependable computing system is one of the potential solutions. In this work, we further extend the *DependableHD* [105] to the second version *DependableHDv2*, which guarantees the operations in sub-threshold voltage regions where even a part of memory cells can not correctly operate due to process variation. We achieve a significant robustness improvement with no extra inference hardware cost. The novelty of our proposed framework is to introduce the concept of margin enhancement during the retraining and utilize random noise injection as well as dimension-swapping techniques to improve the robustness of HDC systems. Such a strategy is capable of application and robustness improvement in most existing state-of-the-art HDC algorithms. The main contributions are listed below:

- Propose margin enhancement and random noise injection techniques to improve the robustness of HDC systems without any extra inference hardware cost.
- To address the stuck errors induced by aggressive voltage scaling in memory cells, a dimension-swapping technique is proposed. By the equivalent structure trans-

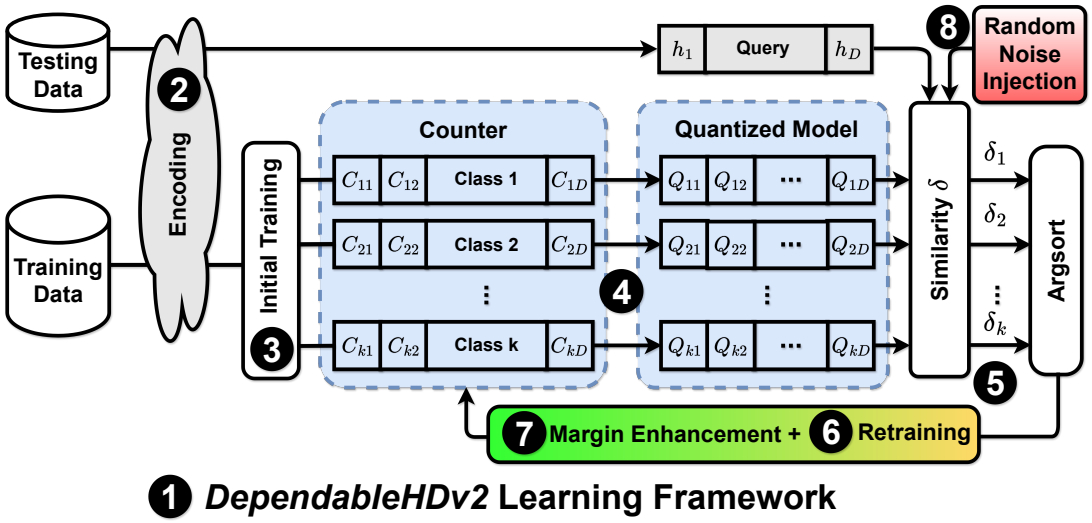

Figure 5.1: Overview of DependableHD (v2) framework.

formation in HDC systems, the valid stuck-at errors can be changed into invalid stuck-at errors, which provides higher robustness for the HDC models.

- We further extend the *DependableHD* framework to the second version, called *DependableHDv2*, which supports the systems to tolerate the serious memory failure induced by aggressive voltage scaling. In addition to the margin enhancement and the random noise injection techniques supported by *DependableHD*, *DependableHDv2* employs a dimension swapping technique which aims at handling the stuck-at errors induced by aggressive voltage scaling in the memory cells. Under the 8% memory stuck-at error rate, the experimental result shows that our proposed HDC framework exhibits a 2.42% accuracy loss on average, which corresponds to a 14.1× robustness improvement.
- The hardware evaluation shows that the supply voltage of our systems can be reduced from 430mV to 340mV for both item Memory and Associative Memory in HDC, which provides a 41.8% energy consumption reduction.

5.2 Proposed Method

In this section, we propose the *DependableHDv2*, which is a combination of three techniques: Fig. 5.1 (❼) margin enhancement, Fig. 5.1 (❽) random noise injection and Fig.5.6 dimension-swapping. The margin enhancement and random noise injection techniques mainly focus on the optimization of the model retraining strategy, which does not require any extra hardware cost during the inference. These two techniques can be simultaneously applied to most state-of-the-art HDC frameworks. They are suitable for both temporary memory errors (e.g., bit-flipping attacks) and permanent memory

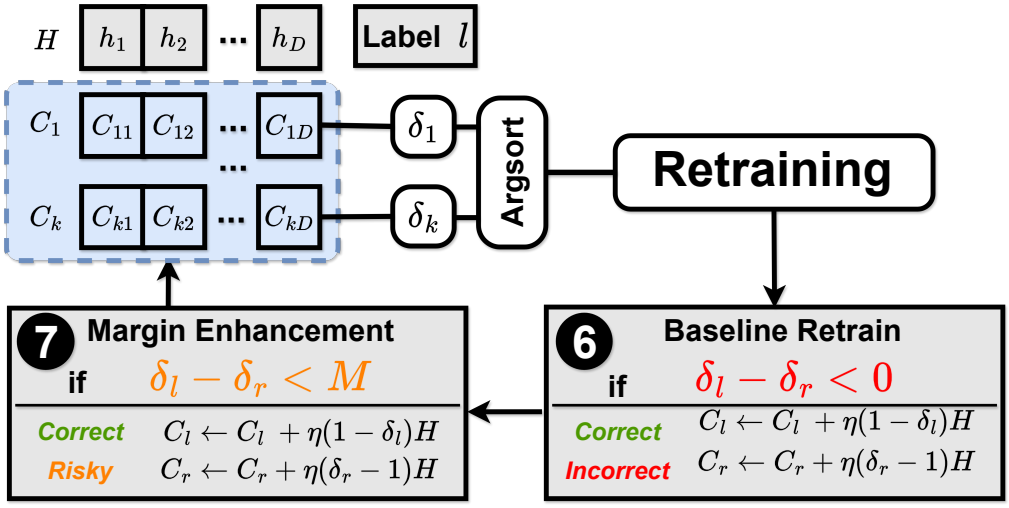

Figure 5.2: Overview of Margin Enhancement technique during Retraining.

errors (e.g., bit cells stuck-at errors), which improves the robustness of HDC models in a universal way. As for the dimension-swapping technique, it particularly aims at handling the stuck-at errors induced by voltage scaling in memory cells.

5.2.1 Margin Enhancement

In HDC classification, the similarity δ_r of target category r is supposed to be the highest one when the prediction is correct. The conventional HDC retraining focuses on updating the model when the prediction is mismatched to another incorrect label l. By adding/subtracting this mismatched training hypervector to the target/predicted class hypervector, the similarity δ_r and δ_l is expected to get higher/lower. Once these two similarity is modified to $\delta_l - \delta_r > 0$, the prediction is successfully recorrected by the traditional retraining. However, the main challenge is that when memory failures arise in the HDC models due to voltage scaling, all the similarity measurements will be affected and result in bias to some extent. Such similarity bias may lead to a change in sorting results, especially for those two key similarities δ_l and δ_r. Once they are offset to $\delta_l - \delta_r < 0$, the prediction is mismatched due to the memory failure and eventually resulting in classification performance loss.

For the retraining in our proposed *DependableHDv2*, instead of naively modifying the mispredictions as Fig.5.2(❻), we propose margin enhancement (❼) which temps to modify the wrong predictions and risky correct predictions. The encoded hypervector of each training sample is created as $\vec{H}$, and then the corresponding similarity with the j-th integer class hypervector $\delta_j = \delta(\vec{H}, \mathbb{Q}^j)$ is checked. Similarly, take δ_l and δ_r as the

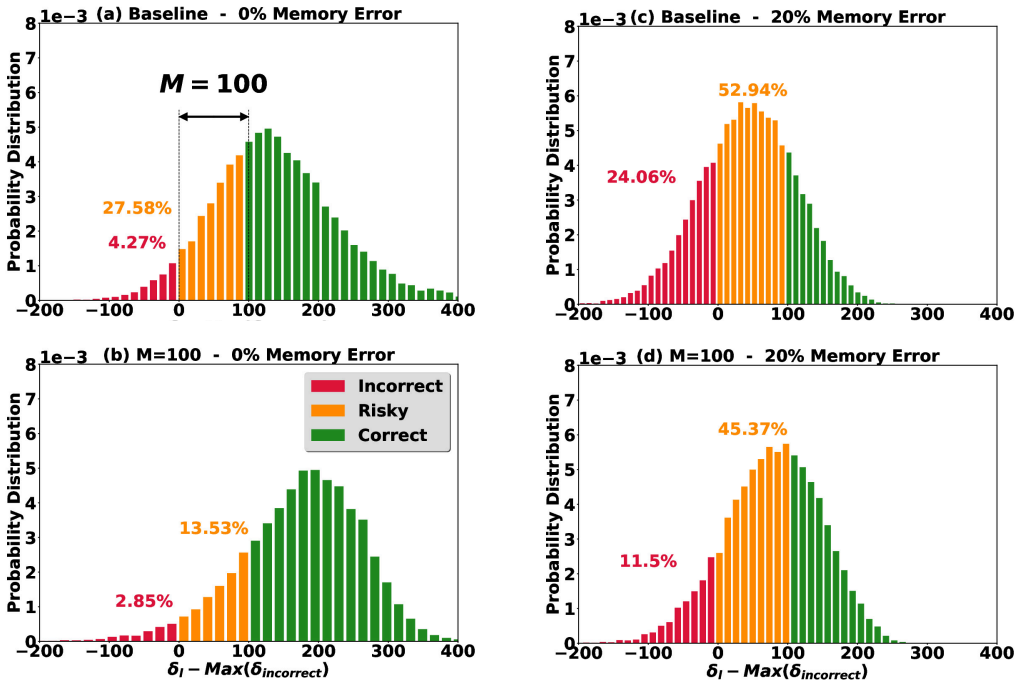

Figure 5.3: Impact of Margin Enhancement technique during Retraining.

similarity for target label l and the highest similarity within all incorrect classes.

$$\delta_l - \delta_r = \delta_l - Max(\delta_{incorrect}). \tag{5.1}$$

We define the margin enhancement level as M for the HDC system with D dimensions. When $0 < \delta_l - \delta_r < M/D$, we consider the distinction for the target label l is insufficient, which means the prediction is risky. In such a case, the counter class hypervectors $\mathbb{C}$ are required to be updated. Mathematically, the retraining process can be performed as Eq. (2.8). In this way, the prediction margin $\delta_l - \delta_r$ tends to enlarge during the retraining phase, which might change the risky predictions into correct and safe predictions.

Figure 5.3 shows an example of baseline binary HDC and applying margin enhancement technique, under 0% and 20% memory error rate, respectively. In this case, both of these two models are utilizing hypervectors with binarized elements, while the Hamming distance is considered as the similarity metric. From Fig. 5.3 (a) and Fig. 5.3 (b), we found that even though the baseline HDC algorithm and *DependableHDv2* achieve similar accuracy under 0% memory error, the probability distribution of prediction margin is obviously changed. Not only improving the average value of the prediction margin but also increasing the standard deviation. Therefore, the number of risky samples is sharped reduced from 27.6% to 13.5%. As shown from Fig. 5.3 (a) to Fig. 5.3 (c), when the memory failure occurs, a large number of risky predictions could be affected and

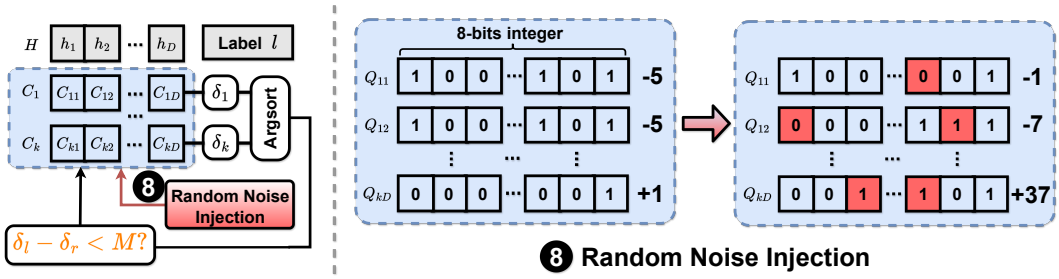

Figure 5.4: Overview of Random Noise Injection technique during Retraining.

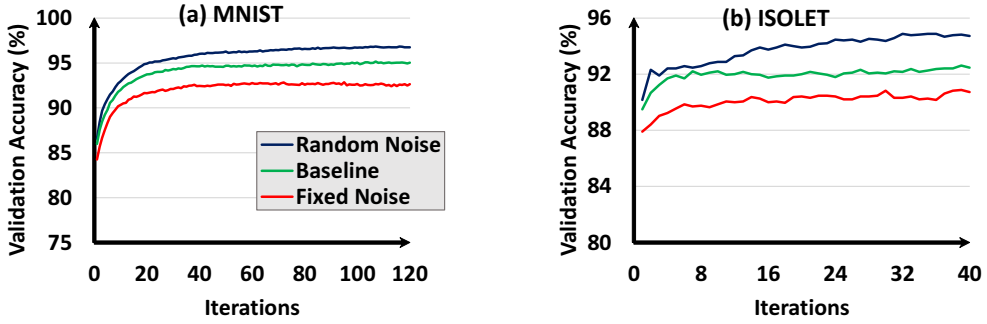

Figure 5.5: Validation Accuracy during iterations.

became misprediction. Hence, the classification accuracy performance sharply drops down from 95.73% to 75.94%. On the other hand, the risky samples are much less in *DependableHDv2* model. Therefore, the bias of similarity results can be tolerated by the HDC system, resulting in a significant accuracy improvement compared with the baseline under 20% memory error rate.

Unlike the previous research [24, 39] focusing on the modification of learning rate, this margin enhancement turns to the optimization of the selected samples during retraining phase. With the utilization of risky samples, the HDC model tends to learn the distinguishing information between risky prediction, resulting in higher robustness against the misprediction induced by memory failures.

5.2.2 Random Noise Injection

Since the memory failures or bit-level attacks arise during the inference for edge-oriented systems, the integer class hypervectors $\mathbb{Q}$ are affected, especially for the most serious bit stuck-at errors. Therefore, we propose random noise injection during the retraining phase, as shown in Fig. 5.4 (❽). Assume the hypervector $\mathbb{Q}$ has D dimensions with m-bits elements, which requires $m \cdot D$ bits of memory to store the information. We define R as the random noise injection level, which represents the percentage of memory stuck-at errors due to random noise injection. Hence, the readout data of $m \cdot D \cdot R$ bits

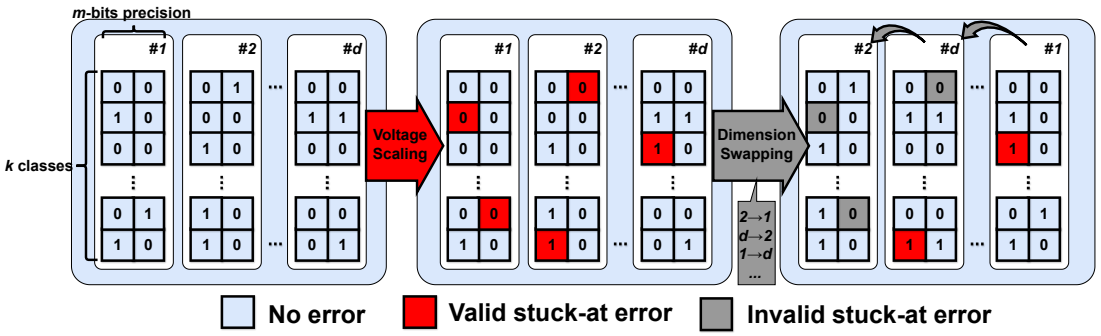

Figure 5.6: Overview of Dimension-swapping technique.

memory are stuck-at logic 1 or logic 0, resulting in various biases of data in the HDC models. For each iteration, the random noise is injected into the ideal class hypervectors independently. This process aims at simulating the similarity bias caused by memory failure when it performs the inference query.

Figure 5.5 shows the validation accuracy performance of applying random noise injection in *DependableHDv2* for MNIST and ISOLET datasets. Similar to the dropout layer in the neural network, this random noise injection technique not only helps the HDC models for robust learning but also addresses the overfitting issue, which contributes to faster convergence and slightly higher saturation accuracy during the retraining.

Note that for improving the robustness against the memory errors in a more general way, the noise is required to be randomly injected into the model for each training epoch. And the memory failure occurrence address needs to be shuffled after each iteration. Otherwise, when injecting the fixed noise into the memory cells, i.e., the positions/addresses of memory cells with flipping errors are fixed, the HDC model data in the corresponding positions can not be effectively modified by iterative learning. The bits with fixed error tend to become the trap for HDC retraining, which results in a slower convergence and poor saturation accuracy.

5.2.3 Dimension-swapping

Due to the utilization of hypervectors, the superior parallelism characteristics provide the HDC model with a unique way to handle the stuck-at errors induced by voltage scaling. Under low voltage operation, SRAM cells may suffer from functional failures (under parameter variations) due to negative read SNM [106] and negative write margin [107]. Since the read SNM and write margin have conflicting design requirements [108], we can ignore the probability that both have negative values simultaneously for the SRAM cells [109]. Applying the aggressive voltage scaling to SRAM cells, the functional failures mainly performs as stuck-at errors, which results in the accuracy performance degradation.

Regardless of the write-in data, the readout data for a memory cell remains stuck-at logic 0 or logic 1. Such memory functional failures can be considered as stuck-at-0 errors and stuck-at-1 errors. When it comes to the implementation of the HDC models, if the ideal write-in data is identical to the readout data, e.g., write-in logic 0 for the memory cells with stuck-at-0 errors, such stuck-at errors are invalid. Otherwise, when the memory cells with stuck-at errors are assigned to store the data with different types, e.g., write-in logic 0 for the memory cells with stuck-at-1 errors, it is considered as valid stuck-at errors, which is the main reason for performance degradation.

Hence, to handle the stuck-at errors induced by voltage scaling in memory cells, our target is to change the valid stuck-at errors into invalid stuck-at errors. As shown in Fig. 5.6, after the assignment of base hypervectors and the training of class hypervectors, the swapping between any two dimensions for all the base hypervectors and class hypervectors can be considered as an equivalent structure transformation. Thanks to such characteristics, we can search the dimension with appropriate ideal data and equivalently swap the data of the corresponding two dimensions, which can partially change the valid stuck-at errors into invalid stuck-at errors.

Assuming we plan to apply the dimension-swapping technique within d dimensions and m-bits precision. The first step is to detect the locations for stuck-at-0 errors and stuck-at-1 errors. We define these locations with stuck-at errors as error matrices E_0 and E_1. Based on the locations of stuck-at errors and the ideal HDC model with d dimensions, we need to determine the new order of dimensions for the HDC model. Each dimension of the ideal HDC model can be considered as the candidate dimension for fitting the stuck-at errors. The dimensions of error matrices with serious stuck-at errors, e.g., high error rate, can be considered as target dimensions with high priority. On the other hand, the dimensions of error matrices with fewer stuck-at errors can be handled by limiting candidate dimensions. Based on such priority, we can swap the corresponding d dimensions of the base hypervectors and the well-trained class hypervectors. Repeat the operations for the next d dimensions until the dimension-swapping is applied to the whole HDC model. After the equivalent structure transformation, the percentage of valid stuck-at errors can be reduced.

Note that the dimension-swapping technique is not suitable for the HDC model [42, 104] with shifting (permutation) operations during the encoding. The detailed hardware implementation of dimension swapping is described in Subsection. 5.3.4.

5.3 Architecture of *DependableHDv2*

The hardware architecture of *DependableHDv2* mainly consists of four different modules: memory blocks, encoding modules, nearest distance searching modules, and dimension-sorting modules.

5.3.1 Memory Blocks

In most HDC frameworks, the item Memory (iM) blocks for hypervectors encoding and the Associative Memory (AM) for the class hypervectors storage are both necessary. For the implementation of iM, the Non-Volatile Memory (NVM), combination circuits, and SRAM circuits are all considered as candidates in the previous work [105]. On the other hand, the AM mainly utilizes SRAM or DRAM for hardware implementation, which depends on the memory capacity requirement. As for *DependableHDv2*, we choose SRAM circuits to implement iM and AM, which share a unified supply voltage source for energy efficiency. Unlike the previous *DependableHD* [105] that only apply the voltage scaling in AM, *DependableHDv2* further applies the voltage scaling to both of these two memory blocks, which brings more challenges for the HDC system robustness requirement. Meanwhile, utilizing SRAM leaves room for the edge-oriented device to apply to multiple tasks.

5.3.2 Encoding Modules

Our approach stores all base hypervectors in the item Memory (iM) blocks. After access to the feature values in the original domain, *DependableHDv2* multiplies each feature value with the base hypervectors. This process can perform in parallel for all features. The results of multiplications are accumulated using D counter blocks, which are then further applied to the Cosine function circuits [110, 111] if the elements are in floating-point format. In this way, an encoded hypervector with D elements as well as corresponding precision is generated.

5.3.3 Nearest Distance Searching

In the inference and dimension-swapping process, the nearest distance searching for similarity measurement is required. For the HDC model with multiple bits integral or floating-point elements, the similarity check is performed using Cosine metric. When the elements are in binary format, the similarity check performs using Hamming metric. Each row computes the distance of a query and class hypervector. A counter/accumulator block is located at the right side of the array, which aims to count/accumulate the number of mismatches in each class. Finally, a tree-based comparator block identifies a class with the minimum distance.

5.3.4 Dimension Sorting Modules

Due to the process variation, the locations of functional failures under low supply voltage depend on chips by chips. For the implementation of dimension-swapping tech-

niques in an offline way, we perform this process in the following steps:

***Step 1*:** Under the target supply voltage, write all logic 1s into the Associative Memory (AM) block for the ideal HDC model with k categories, m-bits precision, and d dimensions. Then store the readout as an error matrix $\mathbb{E}_0 \in \{0,1\}^{k\times m\cdot d}$, which marks the locations of stuck-at-0 errors with logic 0s. Similarly, write all logic 0s into the memory blocks and get the error matrix $\mathbb{E}_1$ for marking the locations of stuck-at-1 errors. Assuming the ideal class hypervectors of the HDC model as $\mathbb{Q} = \{\mathbb{Q}_1, ..., \mathbb{Q}_d\}$, which is also considered as the candidate dimensions.

***Step 2*:** To determine the new order of these d dimensions, we need to calculate the fitting priority of dimensions with errors. Count the number of stuck-at errors for each dimension. With more stuck-at errors in one dimension, more challenging to find the dimension with appropriate memory data that fits the stuck-at errors, which represents a higher priority to be fit.

***Step 3*:** For the dimension with the highest fitting priority, store this index $index^T$ in the list $\mathbb{T}$. Search the dimension with the most appropriate memory data from the candidate dimensions $\mathbb{Q}$, and store its index $index^F$ in the list $\mathbb{F}$.

***Step 4*:** Based on the list $\mathbb{T}$ and $\mathbb{F}$, we got the information that the ideal HDC model should move the $index^F$-th dimension data to the $index^T$-th dimension: $\mathbb{Q}_{index^F} \rightarrow \mathbb{Q}_{index^T}$. Hence, the $index^F$-th dimension should be removed from the candidate dimensions $\mathbb{Q}$, while the $index^T$-th dimension should be removed from the error matrices $\mathbb{E}_0$ and $\mathbb{E}_1$.

***Step 5*:** Repeat *Step* $2 \sim 4$ until the swapping operations for all d dimensions are determined, with the information stored in the list $\mathbb{T}$ and $\mathbb{F}$. Apply the same swapping operations to both the ideal HDC model in AM and the base hypervectors in iM.

In such a dimension-sorting module, the dimension-swapping technique is applied to the d dimensions of the HDC model. This process can perform for all D dimensions sequentially. Note that for the SRAM cells after fabrication, the location of the functional failures are assumed to be fixed under a fixed PVT condition. Hence, under the target supply voltage, we can perform the dimension-sorting modules to change the valid stuck-at errors to invalid stuck-at errors in those fixed SRAM cell locations. Similarly, if the customers are looking for the application of the dimension-swapping technique when facing the functional failures from different temperature, the dimension-sorting modules should be activated under the target temperature condition. By changing the valid stuck-at errors to invalid stuck-at errors, such an equivalent structure transformation helps the model adapt to the specific stuck-at errors under the target condition (supply voltage and temperature) in each chip.

5.4 Experiment

5.4.1 Experimental Setup

The performance and robustness comparison is estimated with six popular datasets: HAR [70], ISOLET [69], MNIST [64], optdigits [112], Fashion-MNIST [98], and Kuzushiji-MNIST [97]. For a comprehensive evaluation of different techniques in our *DependableHDv2*, We firstly study the impact of different key parameters, e.g., the margin enhancement level M, random noise injection level R, and the range of dimension-sorting module d for our *DependableHDv2* system. We evaluate the performance of *DependableHDv2* and the baseline HD algorithm [39] in both software and hardware.

In software, both of these frameworks are written in Python and run on the Intel Core i7 7600 CPU. All the memory failures are set as stuck-at errors, which means the readout data for a memory cell remains stuck-at logic 0 or logic 1. In order to evaluate the performance of HDC model in the low voltage region, the bit cell error patterns of SRAMs in the unit are assigned to be uniformly distributed throughout the memory array in the simulation [78]. The retraining continues for multiple iterations until the validation accuracy has small changes during the last few iterations. We randomly inject the stuck-at errors for 1000 times to explore the robustness of the well-trained models. Note that unlike the previous work [105] only evaluating the functional failures in AM, we are injecting the stuck-at errors into both iM and AM, which brings higher requirement of robustness.

In hardware, we manually design an SRAM with 32×128 bits capacity in 65nm commercial technology using logic process. The functional failure probability of a single SRAM cell are measured based on the bitcell's static noise margin from DC Sweep Analysis, for 10,000 times Monte Carlo simulation using *HSPICE* from *Synopsys* Inc. in a TT (Typical-Typical) condition. The simulation varies device dimensions (gate oxcide thickness, threshold voltage, and so on). We found that writing failure mainly dominates in this process technology. In terms of inference energy consumption, the similarity measure operations are the most dominating, which require the memory readout operation. Therefore, we will use the energy consumption of SRAM for each clock cycle to evaluate the hardware performance of the HDC systems. We utilize testing patterns to sequentially get access to the SRAM, and evaluate the energy consumption based on the post layout netlist utilizing *FineSim* transistor-level circuit simulator from *Synopsys* Inc. in a TT condition.

5.4.2 Impact of Voltage Scaling, Dimensionality and Precision

As shown in Fig. 5.7, the memory error rate sharply increases with the supply voltage downscaling. For a manually designed SRAM in 65nm commercial technology, we

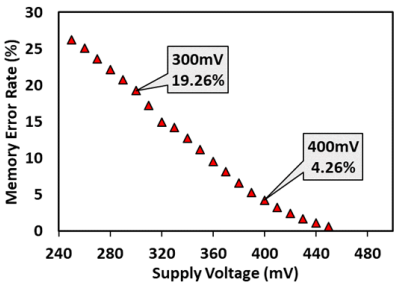

Figure 5.7: Impact of V_{DD}.

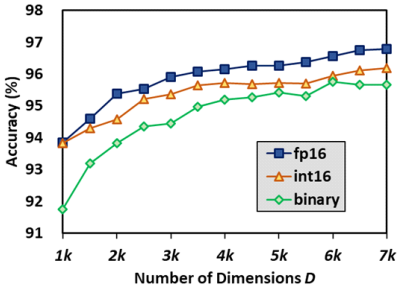

Figure 5.8: Impact of D and precision.

measured the memory error rate of a single SRAM cell based on the bitcell's SNM from DC Sweep Analysis, for 10,000 times Monte Carlo simulation. When the V_{DD} is scaled down to 400mV, a 4.26% memory stuck-at error rate is measured. If we further reduce the V_{DD} to 300mV, the memory error rate explosively increases to 19.26%.

To comprehensively support the requirements for all kinds of platforms and applications, the hypervector representation of HDC systems include fp16, int16, and binary formats. Increasing the number of dimensions or precision of elements results in improving the classification accuracy. However, increasing dimensional also results in an memory capacity and energy efficiency issue. Fig. 5.8 shows the accuracy comparison between the HDC systems with elements in different representations. The results indicate that under the low dimensionality $D = 1K$, the int16 HDC model provides higher accuracy compared to the binary HDC model, while the fp16 model shows the same performance compared to int16 model. When increasing the number of dimension $D = 5-7K$, the binary and int16 model achieves similar maximum accuracy, while the fp16 model provides superior performance.

5.4.3 Margin Enhancement Level

The margin enhancement level (M) significantly impacts the accuracy and robustness of *DependableHD* system. The evaluations of the int16 model for MNIST dataset with $D = 3K$, is shown in Fig. 5.9. During the experiment, the stuck-at errors are randomly injected into all the memory blocks, including both the item Memory block for base hypervectors and the Associated Memory block for class hypervectors.

On one hand, with the increase of M in the range of $0 \sim 120$, the saturation accuracy is slightly improved. The enhancement of prediction in risky samples benefits the retraining of *DependableHDv2* model. On the other hand, when the M is too large, the number of samples treated as risky explosively grows. And the percentage of misprediction samples that can update the model is declined, which may result in overfitting issues. Hence, there is a trade-off between the robustness and convergence accuracy performance promoted by this margin enhancement technique.

In terms of robustness, using a large M represents using more samples with risky

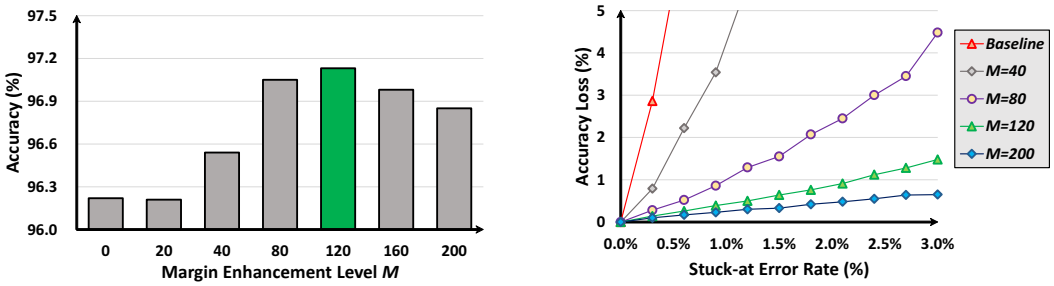

Figure 5.9: Impact of Margin Enhancement Level M.

predictions for retraining the model. This results in the expansion of the prediction margin, which significantly reduces the percentage of risky predictions and eventually contributes to the decrease in accuracy loss. The margin enhancement technique is capable of the HDC model with different element precision, which is shown in Table. 5.2.

5.4.4 Random Noise Injection Level

Figure. 5.10 shows the impact of random noise injection level (R) for MNIST dataset in int16 elements precision with $D = 3K$. During the training and inference phase, the stuck-at errors are randomly injected into all the memory blocks, including both the iM block and the AM block.

Similar to the dropout layer in the neural network, this random noise injection technique not only helps the HDC models for robust learning. With the increase of R in the range of $0 \sim 8\%$, the random noise addresses the overfitting issue for HDC models, which contributes to a slight improvement in saturation accuracy. However, when the R is too large, the encoded hypervectors are polluted by the injected noise during the retraining, which limits the maximum accuracy. Hence, there is also a trade-off between the robustness and convergence accuracy.

In terms of robustness, using a large R can simulate serious functional failure during the retraining, which helps the model to adapt the functional failure is a general and universal way. As shown in Fig. 5.10, the robustness of HDC system is improved with the increase of R. Note that both margin enhancement and random noise injection tech-

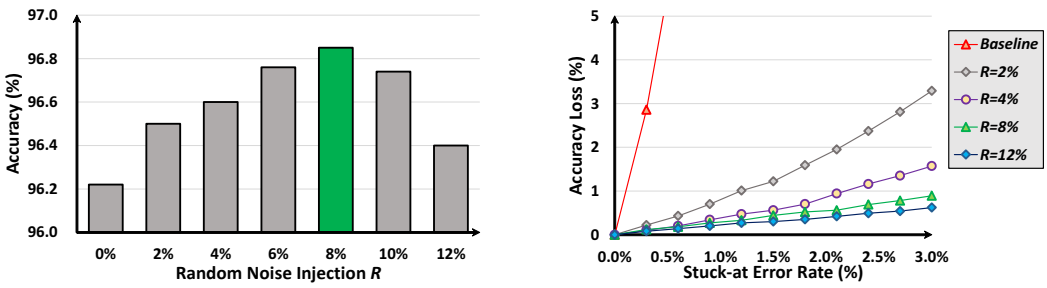

Figure 5.10: Impact of Random Noise Injection Level R.

niques aims at exploring the robustness of different HDC models. Hence, the optimal rate of random noise injection also depends on the dataset, model size (dimensionality of hypervectors), element precision, and margin enhancement level M. The random noise injection technique is also capable of the HDC model with different element precision, which is shown in Table. 5.2 in detail.

5.4.5 Performance of Dimension-swapping

Figure. 5.11 shows the performance of dimension-swapping techniques. This technique aims at changing the valid stuck-at errors into invalid stuck-at errors. The performance of the dimension-swapping technique is determined by two parameters, the range of dimension-sorting module d and the number of stuck-at errors in each dimension.

Take the binary *DependableHDv2* model with $D = 10K$ for the MNIST dataset as an example, when the stuck-at errors are randomly injected into the AM block, we apply the dimension-swapping techniques. Fig. 5.11 (a) shows the reduction of valid stuck-at error rate achieved by different ranges of dimension-sorting d. With the increase of dimension-sorting range d, each dimension-sorting module contains more candidate dimensions to fit the stuck-at errors. The sufficient candidate dimensions can significantly reduce the valid stuck-at error rate. When $d = 32$, the percentage of valid stuck-at errors sharply decreases from 10% to 2.54%, which achieved a 74.6% reduction. Meanwhile, when the number of stuck-at errors in each dimension is increasing, it gets more challenging to find the appropriate candidate dimensions to fit the stuck-at errors. With the same range $d = 32$, the original 20% stuck-at error rate was only reduced to 9.06% after applying dimension-swapping technique, which is a 54.7% reduction and results in the accuracy improvement as shown in Fig. 5.11 (b).

Note that number of stuck-at errors in each dimension differs with the applications. For example, if the target task is changed from MNSIT dataset with 10 categories to ISOLET dataset with 26 categories, the number of stuck-at errors in AM block also increases. If the representation of element is changed from binary to int16 format, the number of stuck-at errors in each dimension is expected to increase by 16×, which could

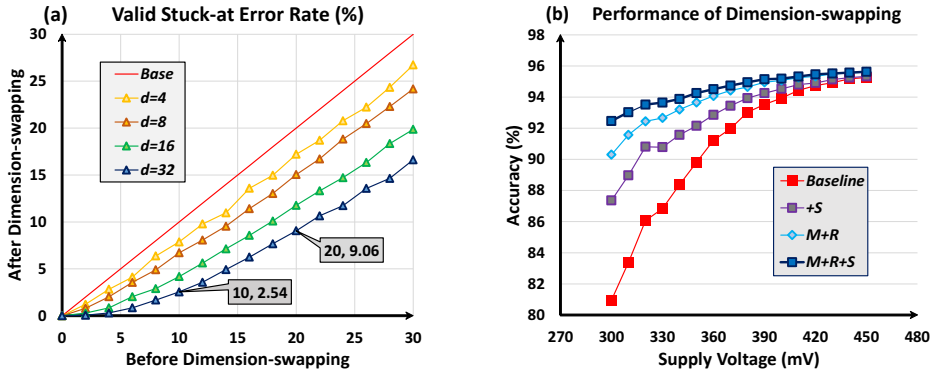

Figure 5.11: Impact of Dimension-swapping technique.

Table 5.1: *DependableHDv2* Parameters Setup.

	mnist, optdigits			**isolet**			**har**			**k&fmnist**
	fp16	**int16**	**bin**	**fp16**	**int16**	**bin**	**fp16**	**int16**	**bin**	**bin**
D	3K	3K	10K	3K	3K	10K	3K	3K	10K	20K
+M	120	120	240	200	200	240	100	100	60	240
+R	6%	8%	6%	1%	1%	0.5%	4%	4%	2%	6%
+S	–	–	16	–	–	16	–	–	16	16

affect the valid stuck-at error rate reduction. Hence, the dimension-swapping technique mainly focuses on optimization for the binary HDC models.

The dimension-swapping technique can be achieved both online and offline. To evaluate the hardware overhead when users require to perform in edge-oriented devices, we use *Synopsys Design Compiler* to synthesize and report the area consumption of our approach in a 65-nm ASIC flow. The hardware cost of memory (iM and Am) was individually simulated using *CACTI*, which is an integrated memory access time, area, leakage, and power model. Since the dimension-swapping operations are performed before writing the well-trained *DependableHDv2* model into voltage-scaled devices for once, the area overhead becomes the main concern. For the binary *DependableHDv2* model with $D = 10K$, the area overhead consumed by the dimension-sorting module with $d = 32$ is 3.04% compared with the AM block, which is negligible compared with the total memory blocks ($< 1\%$). Though a larger d requires sorting within a wider range and increases the computational cost for the training, it is still considered a practical implementation for edge-oriented devices.

5.4.6 *DependableHDv2* Robustness

As observed in Subsections 5.4.3 and 5.4.4, when taking the margin enhancement level M and random noise injection level R, there is a trade-off between the convergence accuracy performance and the model robustness. The dimension-sorting range d also affects the performance of the dimension-swapping technique. For the *DependableHDv2* implementation, we choose the compromised parameter for competitive accuracy and sufficient robustness, which is shown in Table. 5.1. In terms of the training cost, the *DependableHDv2* requires $1.18\times$ training time compared with the baseline on average. Since the training process can be individually performed on the cloud, a slight increase in training cost is acceptable for edge-oriented systems.

The comparison of robustness is shown in Table 5.2 (for MNIST, ISOLET, and HAR datasets). We comprehensively evaluate the robustness improvement from three key techniques: **M**argin enhancement, **R**andom noise injection, and dimension-**S**wapping. Our experiment shows that under the 8% memory error rate, the *DependableHDv2* exhibits 2.42% accuracy loss on average, which achieves a $14.1\times$ robustness improvement compared to the baseline HDC solution.

Table 5.2: Accuracy Loss Comparison.

	MNIST				**ISOLET**				**HAR**			
Acc Loss	**Stuck-at Error Rate**				**Stuck-at Error Rate**				**Stuck-at Error Rate**			
(%)	***2%***	***4%***	***6%***	***8%***	***2%***	***4%***	***6%***	***8%***	***2%***	***4%***	***6%***	***8%***
FP16 Base	13.21	24.17	30.84	37.88	21.90	25.27	28.67	29.18	23.57	30.00	41.21	46.30
+M	0.82	2.20	4.12	6.96	2.56	6.27	8.47	11.04	0.59	1.01	3.74	5.03
+R	0.70	1.79	3.45	5.73	0.81	2.95	5.15	8.05	0.25	0.65	2.07	4.17
M+R	0.39	0.87	1.47	2.34	0.74	1.57	2.45	3.43	0.78	0.91	1.65	2.90
	2%	***4%***	***6%***	***8%***	***2%***	***4%***	***6%***	***8%***	***2%***	***4%***	***6%***	***8%***
Int16 Base	24.39	41.96	50.94	57.63	42.39	51.63	57.51	61.97	27.08	48.73	53.26	56.15
+M	0.85	2.16	4.28	6.93	5.77	12.38	19.04	21.65	1.02	4.79	9.49	23.37
+R	0.53	1.29	2.17	3.73	3.51	9.18	14.13	19.82	0.73	1.01	2.26	2.70
+M+R	0.33	0.68	1.07	1.55	0.53	1.27	2.24	4.13	0.26	0.70	1.21	2.08
	4%	***8%***	***12%***	***16%***	***4%***	***8%***	***12%***	***16%***	***4%***	***8%***	***12%***	***16%***
Binary Base	1.27	3.63	6.60	11.41	1.05	1.81	2.58	3.72	5.50	12.43	16.08	22.62
+M	0.73	1.57	3.22	5.48	0.25	0.48	1.28	1.75	4.80	9.77	12.81	19.82
+R	0.86	2.29	4.02	6.63	1.09	1.85	2.54	3.73	4.57	9.58	8.84	17.54
M+R	0.64	1.30	2.43	3.93	0.69	1.11	1.75	2.49	1.76	5.58	7.38	10.44
+S	0.84	2.13	3.55	5.66	0.59	0.91	1.29	1.71	3.06	6.07	6.68	9.02
M+R+S	0.44	1.06	1.58	2.39	0.53	0.83	1.06	1.47	1.40	3.49	4.23	5.15

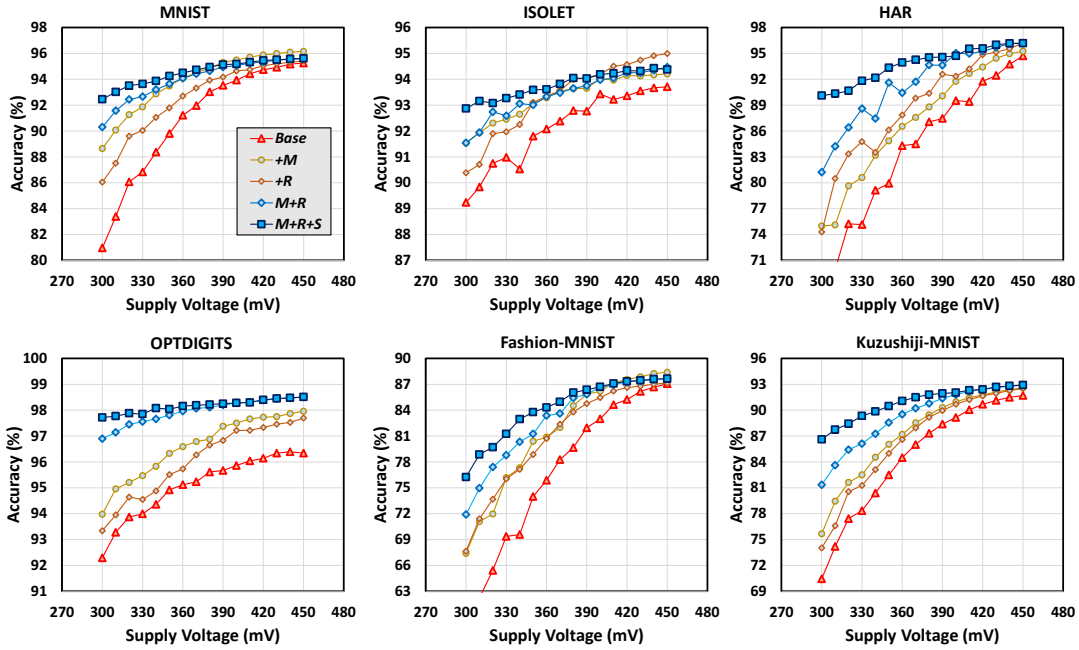

Figure 5.12: Accuracy of Binary *DependableHDv2* under different Supply Voltages.

5.4.7 Energy Consumption Reduction

Fig. 5.12 shows the comparison of performance under different supply voltages. *DependableHD* utilizes the margin enhancement and the random noise injection techniques to improve the robustness of the Baseline HD. *DependableHDv2* is an improved version of *DependableHD* employing the dimnesion swapping technique. Our evaluation shows that *DependableHDv2* averagely provides 91.73% accuracy when the supply voltage of SRAM (including AM and iM) is downscaled to 340mV, which is significantly higher performance as compared to the 83.73% accuracy achieved by the baseline HDC model [39] and the 89.82% accuracy achieved by *DependableHD* [105]. All three techniques (margin enhancement, random noise injection, and dimension-swapping) contribute to robustness improvement.

Note that compared with the previous work [105], the *DependableHDv2* supports the application of voltage scaling in item Memory (iM), which usually consumes much more capacity compared with the Associative Memory [96]. Though the functional failures in the iM bring the challenges of accuracy degradation, the total energy reduction guarantees the edge-oriented system's long-time operation and high energy efficiency. As a combination of margin enhancement, random noise injection, and dimension-swapping, the performance of *DependableHDv2* is further improved. We utilize the readout energy consumption per clock cycle of a 32x128 bits SRAM to represent the energy consumption for HDC models. As shown in Table 5.3, downscaling the supply voltage from 430mV to 370mV and 340mV can provide a 29.2% and a 41.8% energy

Table 5.3: Energy Consumption Comparison.

	$@V_{dd}$	**Accuracy**	**Energy Consumption**	**Energy Reduction**
Baseline HD [39]	430mV	92.42%	0.93pJ	–
***DependableHD* [105]**	370mV	91.92%	0.66pJ	29.2%
DependableHDv2	340mV	91.73%	0.54pJ	41.8%

consumption reduction, respectively. A significant energy consumption reduction is achieved for the inference phase.

5.5 Conclusion

This paper extends the novel HDC learning framework *DependableHD* [105] to the second version *DependableHDv2*, which supports the systems to tolerate the memory functional failures in the low voltage region with high robustness. After statistically analyzing the impact of memory failures for HDC systems, we introduce the concept of margin enhancement during the model retraining. Instead of the traditional retraining aims at correcting the missed predictions, our strategy also targets enhancing the distinction when the predictions are correct but risky. Meanwhile, we propose random noise injection during the model training for more robust learning. Based on the high parallelism of HDC, we propose a dimension-swapping technique to handle the stuck-at errors induced by voltage scaling in memory cells, which aims at changing the valid stuck-at errors into invalid stuck-at errors.

Under an 8% memory error rate, the *DependableHDv2* exhibits a 2.42% accuracy loss on average, which corresponds to a 14.1× robustness improvement compared to the state-of-the-art HDC system. The sufficient robustness enables the systems to operate even after scaling down the supply voltage from 430mV to 340mV, with negligible accuracy loss. The aggressive voltage scaling strategy are applied to both item Memory and Associative Memory in HDC systems. The hardware evaluation shows that the reduction of supply voltage provides a 41.8% energy consumption reduction. Such a robust and energy efficient learning framework is capable of improvement and application in most existing state-of-the-art HDC algorithms without any extra inference hardware cost, which equivalently guarantees the improvement of energy efficiency comes from the near-threshold voltage computing.

Chapter 5 contains material from “DependableHD: A Hyperdimensional Learning Framework for Edge-oriented Voltage-scaled Circuits”, by Dehua Liang, Hiromitsu Awano, Noriyuki Miura, and Jun Shiomi, which appears in Proceedings of the 28th Asia and South Pacific Design Automation Conference (ASP-DAC), January 2023 [105].

The dissertation author was the primary investigator and author of this paper.

Chapter 5 contains material from "A Robust and Energy Efficient Hyperdimensional Computing System for Voltage-scaled Circuits", by Dehua Liang, Hiromitsu Awano, Noriyuki Miura, and Jun Shiomi, which appears in ACM Transactions on Embedded Computing Systems, September 2023 [113]. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Summary

6.1 Summary of This Thesis

With the emergence of the Internet of Things (IoT), devices are generating massive data streams. Running big data processing algorithms, e.g., machine learning, on edge devices poses substantial technical challenges due to limited device resources. Compared to the sophisticated machine learning method, the brain-inspired high-dimensional computing paradigm are considered as promising alternative in terms of energy efficiency and robustness, which is suitable for the resource limited scenario. As a novel computing paradigm, how to reduce the hardware cost while maintaining sufficient accuracy performance on edge devices is still an open problems for circuit designers. The goal of this thesis is thus to provide the strategies to design an energy efficient high-dimensional computing paradigm on edge devices. In this thesis, we provide the solutions for high-dimensional computing from different perspectives: arithmetic operations, memory usage, and robustness, which eventually leading to the pursuit of energy efficiency.

In Chapter 3, we discussed the bottleneck for the reservoir computing (RC) systems in several related papers, which is one of the typical high-dimensional computing paradigms. Suffered from the huge memory usage and expensive arithmetic operations, there was still gap between the RC concept to practical implementation on edge devices. Therefore, this thesis propose a novel RC architecture *EnsembleBloomCA*, which utilizes cellular automata (CA) and an ensemble Bloom filter to organize an RC system. By adopting CA as the reservoir in the RC system, it can be implemented using only binary operations and is thus energy efficient. The rich pattern dynamics created by CA can map the original input into a high-dimensional space and provide more features for the classifier. Applying the ensemble Bloom filters as the classifier, the features provided by the reservoir can be effectively memorized. As the combination of these two techniques, the novel RC architecture successfully eliminates all floating-point calculation and integer multiplication. Our experiment result demonstrated that 43$\times$ and

8.5× reduction is achieved in terms of memory usage and power consumption, while the accuracy performance is maintained.

Although the extreme energy efficiency is achieved in the first prototype with novel RC architecture, the circuits designer's target changes for variety application scenarios. The high-dimensional computing paradigm are also expected to achieve not only sufficient but also competitive accuracy performance, which has posed designers problems to balance the trade-off between different key indicators. In Chapter 4, we found the similar issues should be tackled in hyper-dimensional computing (HDC), which are also considered as high-dimensional computing paradigm. To alleviate the huge memory cost during encoding procedure in HDC (i.e., over 95% of the memory capacity is consumed), we propose a novel HDC architecture *StrideHD* that utilizes the window striding in image classification. It encodes data points to distributed binary hypervectors and eliminates the expensive Channel item Memory (CiM) and item Memory (iM) in the encoder, which significantly reduces the required hardware cost for inference. For the improvement of accuracy on edge devices during inference, we provide the iterative learning mode for the proposed HDC architecture. It also enables HDC systems to be trained and tested using binary hypervectors and achieves very competitive accuracy performance. Our experiment result shows that the proposed HDC model achieves extreme memory efficiency (27.6×) with acceptable accuracy performance under the single-pass mode, while its iterative mode provides competitive performance (11.33% classification accuracy improvement) and keeping the memory efficiency (8.7*times* improvement) for inference phase. The iterative retraining can be accomplished within fewer iterations compared to the baseline HDC works.

Besides the trade-off between accuracy performance and hardware resource in high-dimensional computing, the robustness issue are also arousing the attention of circuit designers. As we commonly known, scaling down the supply voltage is a promising approach to reduce the energy consumption of the circuits, while the aggressive voltage scaling will pose designers several severe problems such as the performance variation and functional failures. It is a potential solution to tolerate the voltage-scaling induced functional failures by the superior robustness of the brain-inspired high-dimensional computing paradigms. Therefore, we introduces the concept of margin enhancement for model retraining and utilizes noise injection to improve the robustness in our proposed HDC framework *DependableHD*, which is capable of application in most state-of-the-art HDC algorithms. After analyzing the error patterns in voltage-scaled circuits, we come up with a strategy to take fully advantage of the equivalent structure transformation in HDC architecture. We additionally propose the dimension-swapping technique, which aims at handling the stuck-at errors induced by aggressive voltage scaling in the memory cells. Our experiment shows that under 8% memory stuck-at error, the proposed method exhibits a 2.42% accuracy loss on average, which achieves a 14.1× ro-

bustness improvement compared to the baseline HDC solution. Our work also supports the systems to reduce the supply voltage from 430mV to 340mV for both item Memory and Associative Memory, which provides a 41.8% energy consumption reduction while maintaining competitive accuracy performance.

6.2 Clarification of Proposed Methods

Prior research on DNNs reports that the energy consumption of the system with AI application mainly comes from two perspectives: the memory and processing units. On one hand, the scale of DNN models is sharply increasing. Table 6.1 shows the memory requirement of the popular DNN models, while the last column represents the accuracy performance of each DNN model in MNIST task. Millions of parameter need to be trained and stored in the memory (e.g., off-chip DRAM), while the data movement usually dominates the energy consumption. On the other hand, DNNs require massive numbers of multiply-accumulate (MAC) operations, which leads to a huge amount of dedicated hardware resources and incurs a large energy consumption. To alleviate the expensive energy consumption of the data movement and MAC operations, many researches propose DNN accelerators utilizing computing-in-memory (CIM) macro architectures to improve energy efficiency and maintain accuracy performance. Table 6.2 shows the hardware cost comparison between DNN accelerators and our proposed architectures. Here, we roughly assume the MobileNet V3 Small model are fully implemented by different CIM macro accelerators, while impact in power and area related to CMOS process is proportional. The accuracy performance comparison is based on MNIST dataset. Thanks to the light-weight model and hardware-friendly arithmetic operations in high-dimensional computing, our proposed *EnsembleBloomCA* and *StrideHD* shows competitive performance in terms of power and area.

Table 6.1: Computational Characteristics and Accuracy Performance of Popular DNN Models.

Models	**Parameters (M)**	**Memory (MB)**	**Accuracy (%)**
VGG 16 [114]	138	527.8	99.04
AlexNet [5]	61.1	233.1	98.71
Resnet 18 [115]	11.7	44.7	99.05
EfficientNet B_0 [116]	5.29	20.5	97.84
MobileNet V3 Small [117]	**2.54**	**9.8**	**98.35**
MNASNet 0.5 [118]	2.22	8.6	93.55
EnsembleBloomCA	**0.147**	**0.018**	**91.86**
StrideHD	**0.789**	**0.096**	**97.51**

Table 6.2: Hardware Cost Comparison between DNN Accelerators and our Proposed Architectures.

Architecture	Process	CIM Macro			Total @(MobileNet, MNIST)		
		Power	Area	Memory	Power	Area	Accuracy
	(nm)	(mW)	(mm^2)	(KB)	(W)	(mm^2)	(%)
ISSCC'21 [119]	5	327	5.46	3072	32.71*	546.3*	
ISSCC'24 [120]	14	4300	-	3000	16.2*	-	
ISSCC'24 [121]	28	5.5	1.41	16	3.369	853.6	
ISSCC'19 [122]	8	39	5.5	1568	2.989*	421.2*	98.35%
ISSCC'24 [123]	28	98	3.8	513	1.872	72.59	
ISSCC'21 [124]	28	19.4	1.9	206	0.923	90.39	
ISSCC'24 [125]	22	13.13	8.2	8000	0.026*	16.28*	
EnsembleBloomCA	65	-	-	-	0.029*	0.065*	91.86%
StrideHD	65	-	-	-	0.025*	0.598*	97.51%

*Assuming a proportional impact in power and area related to CMOS process.

Summarizing the discussions so far, suitable edge-computing scenarios of the proposed methods are clarified as follows:

- *EnsembleBloomCA*: Simple classification task. Take MNIST dataset as an example, the proposed *EnsembleBloomCA* model can successfully eliminate all the FP/integer arithmetic operations, supporting the hardware-friendly bit-wise operations for extreme energy efficiency. With around 18 KB memory usage, 91% classification accuracy is achieved. Compared to existing methods, *EnsembleBloomCA* provides a 27.6× memory efficiency improvement. In conclusion, this architecture and this design strategy are suitable for the scenarios with extremely strict resource constrains.
- *StrideHD*: Medium task. This architecture supports element in fp16, int16, and binary format. The wide range of element precision provides sufficient room for the customers to select the appropriate strategy. The single pass mode training can achieve acceptable accuracy performance, while the iterative training mode can further provide 13% accuracy improvement. Even for the complicated dataset like K-MNIST and F-MNIST, competitive accuracy performance is achieved while maintaining 8.7× memory efficiency. In conclusion, this architecture and this design strategy are suitable for the scenarios with limited resource and facing complex datasets.
- *DepedableHDv2*: Medium task under unreliable environments. Take MNIST dadaset as an exmaple, the proposed *DepedableHDv2* model can tolerate more than a 10% memory error. Exploiting this advantage, aggresive voltage down-

scaling can be applied to HDC systems. As a result, the energy efficiency can be improved by 41.8% compared with exisitng methods. Note that both of margin enhancement and random noise injection techniques in this model are focusing on the retraining phase, which means it can be easily applied to other HDC architectures simultaneously for robustness improvement. In conclusion, this architecture and this design strategy are suitable for the scenarios with a extremely unreliable and noisy situation.

6.3 Future Works

Currently, our three proposed architectures can be applied to different scenarios individually. If we are looking for the combination between the proposed methods, the key idea of *DependableHDv2* can be expected to apply to the *EnsembleBloomCA* architecture or *StrideHD* architecture simultaneously.

On one hand, the current *EnsembleBloomCA* architecture mainly focus on the single-pass training mode. For the further improvement of accuracy performance, it is a potential solution to extend the single-pass mode training method to the iterative mode. Since the key idea of *DependableHDv2* locates on the optimization of traditional iterative retraining process. Applying the retaining strategy from *DependableHD*, *i.e.*, margin enhancement and random noise injection techniques, to the current *EnsembleBloomCA* architecture, the accuracy performance and model robustness are expected to be improved. Note that such application of retraining strategy doesn't require any hardware cost for the inference phase.

On the other hand, the current *StrideHD* architecture shares similar iterative retraining strategy with other HDC models. It requires much less effort for the combination of these two methods. With the utilization of hardware-friendly pseudo random hypervector generator during encoding phase and robustness improvement during retraining phase, the combination of *StrideHD* and *DependableHDv2* is expected to achieve competitive accuracy, high energy efficiency, and superior robustness.

Bibliography

[1] "Statista, artificial intelligence (ai) market size worldwide in 2021 with a forecast until 2030 (in million u.s. dollars)," https://www.statista.com/statistics/1365145/artificial-intelligence-market-size/, accessed: 2024-02-28.

[2] F. H. Sinz, X. Pitkow, J. Reimer, M. Bethge, and A. S. Tolias, "Engineering a less artificial intelligence," *Neuron*, vol. 103, no. 6, pp. 967–979, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0896627319307408

[3] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[4] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn, "Machine learning model sizes and the parameter gap," 2022.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[6] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *ArXiv*, vol. abs/1906.02243, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:174802812

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016.

[8] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16. [Online]. Available: https://doi.org/10.1145/2342509.2342513

[10] C.-Y. Chang, Y.-C. Chuang, C.-T. Huang, and A.-Y. Wu, "Recent progress and development of hyperdimensional computing (hdc) for edge intelligence," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2023.

[11] A. Bellaouar and M. Elmasry, *Low-power digital VLSI design: circuits and systems*. Springer Science & Business Media, 2012.

[12] cpudb.stanford.edu, "CPU database [Online]."

[13] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "Cpu db: recording microprocessor history," *Commun. ACM*, vol. 55, no. 4, p. 55–63, apr 2012. [Online]. Available: https://doi.org/10.1145/2133806.2133822

[14] P. A. Gargini, "How to successfully overcome inflection points, or long live moore's law," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 51–62, 2017.

[15] L. Xiu, "Time moore: Exploiting moore's law from the perspective of time," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 1, pp. 39–55, 2019.

[16] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[17] W. M. Holt, "1.1 moore's law: A path going forward," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 8–13.

[18] P. Gargini, ""roadmap evolution: From ntrs to itrs, from itrs 2.0 to irds," in *2017 Fifth Berkeley Symposium on Energy Efficient Electronic Systems & Steep Transistors Workshop (E3S)*, 2017, pp. 1–62.

[19] L. Kish, "Moore's law and the energy requirement of computing versus performance," *Circuits, Devices and Systems, IEE Proceedings* -, vol. 151, pp. 190 – 194, 05 2004.

[20] T. M. Conte, E. P. DeBenedictis, P. A. Gargini, and E. Track, "Rebooting computing: The road ahead," *Computer*, vol. 50, no. 01, pp. 20–29, jan 2017.

[21] C. Jun, "Proactive supply noise mitigation and design methodology for robust vlsi power distribution," PhD thesis, Osaka University, Osaka, January 2020, available at https://ir.library.osaka-u.ac.jp/repo/ouka/all/76645/31287_Dissertation.pdf.

[22] K. Y. Chan, B. Abu-Salih, R. Qaddoura, A. M. Al-Zoubi, V. Palade, D.-S. Pham, J. D. Ser, and K. Muhammad, "Deep neural networks in the cloud: Review, applications, challenges and research directions," *Neurocomputing*, vol. 545, p. 126327, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231223004502

[23] D. Xu, X. He, T. Su, and Z. Wang, "A survey on deep neural network partition over cloud, edge and end devices," 2023.

[24] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 126–131.

[25] M. Cucchi, S. Abreu, G. Ciccone, D. Brunner, and H. Kleemann, "Hands-on reservoir computing: a tutorial for practical implementation," *Neuromorph. Comput. Eng.*, vol. 2, no. 3, p. 32002, 2022. [Online]. Available: https://doi.org/10.1088/2634-4386/ac7db7

[26] R. Shulman, F. Hyder, and D. Rothman, "Energetic basis of brain activity: Implications for neuroimaging," *Quarterly reviews of biophysics*, vol. 35, pp. 287–325, 09 2002.

[27] M. Fox and M. Raichle, "Spontaneous fluctuations in brain activity observed with functional magnetic resonance imaging," *Nature reviews. Neuroscience*, vol. 8, pp. 700–11, 10 2007.

[28] B. Babadi and H. Sompolinsky, "Sparseness and expansion in sensory representations," *Neuron*, vol. 83, no. 5, pp. 1213–1226, 2014.

[29] H. Zhang and D. V. Vargas, "A survey on reservoir computing and its interdisciplinary applications beyond traditional machine learning," *IEEE Access*, vol. 11, p. 81033–81070, 2023. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2023.3299296

[30] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*, 2007, pp. 471–482.

[31] M. Lukosevicius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, pp. 127–149, 2009.

[32] J. Shiomi, "Performance modeling and on-chip memory structures for minimum energy operation in voltage-scaled lsi circuits," PhD thesis, University of Kyoto, Kyoto, October 2017, available at https://repository.kulib.kyoto-u.ac.jp/dspace/bitstream/2433/228252/2/djohk00658.pdf.

[33] S. Wu, H. R. Zhang, and C. Ré, "Understanding and improving information transfer in multi-task learning," 2020. [Online]. Available: https://arxiv.org/abs/2005.00944

[34] A. Morán, C. F. Frasser, and J. L. Rosselló, "Reservoir computing hardware with cellular automata," *ArXiv*, vol. abs/1806.04932, 2018.

[35] A. Lopez, J. Yu, and M. Hashimoto, "Low-cost reservoir computing using cellular automata and random forests," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[36] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[37] M. Imani, Y. Kim, M. S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *IEEE Cloud Computing (CLOUD)*, IEEE. IEEE, 08/2019 2019.

[38] K. Behnam, X. Hanyang, M. Justin, and R. Tajana, "tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications," in *IEEE/ACM Design Automation and Test in Europe Conference (DATE)*, IEEE. IEEE, 2021.

[39] A. Hernandez-Cane, N. Matsumoto, E. Ping, and M. Imani, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 56–61.

[40] M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, and T. Rosing, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 190–198.

[41] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[42] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, 2016, pp. 64–69.

[43] L. Santiago, L. Verona, F. Rangel, F. Firmino, D. Menasché, W. Caarls, M. Breternitz, S. Kundu, P. Lima, and F. M. G. França, "Weightless neural networks as memory segmented bloom filters," *Neurocomputing*, 2020.

[44] L. S. de Araújo, L. Verona, F. Rangel, F. F. de Faria, D. Menasché, W. Caarls, M. Breternitz, S. Kundu, P. Lima, and F. França, "Memory efficient weightless neural network using bloom filter," in *ESANN*, 2019.

[45] Y. Kim, M. Imani, and T. S. Rosing, "Efficient human activity recognition using hyperdimensional computing," in *Proceedings of the 8th International Conference on the Internet of Things*, 2018, pp. 1–6.

[46] S. Jain, S. Khare, S. Yada, V. Ambili, P. Salihundam, S. Ramani, S. Muthukumar, M. Srinivasan, A. Kumar, S. K. Gb, R. Ramanarayanan, V. Erraguntla, J. Howard, S. Vangal, S. Dighe, G. Ruhl, P. Aseron, H. Wilson, N. Borkar, V. De, and S. Borkar, "A 280mv-to-1.2v wide-operating-range ia-32 processor in 32nm cmos," in *2012 IEEE International Solid-State Circuits Conference*, 2012, pp. 66–68.

[47] B. Calhoun and A. Chandrakasan, "Characterizing and Modeling Minimum Energy Operation for Subthreshold Circuits," in *International Symposium on Low Power Electronics and Design*, Aug. 2004, pp. 90–95.

[48] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "Near-threshold voltage (ntv) design — opportunities and challenges," in *DAC Design Automation Conference 2012*, 2012, pp. 1149–1154.

[49] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

[50] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional computing as a nanoscalable paradigm," *IEEE*

Transactions on Circuits and Systems I: Regular Papers, vol. 64, no. 9, pp. 2508–2521, 2017.

[51] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 445–456.

[52] T. F. Wu, H. Li, P.-C. Huang, A. Rahimi, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 492–494.

[53] H. Li, T. F. Wu, S. Mitra, and H.-S. P. Wong, "Device-architecture co-design for hyperdimensional computing with 3d vertical resistive switching random access memory (3d vrram)," in *VLSI Technology, Systems and Application (VLSI-TSA), 2017 International Symposium on*, IEEE. IEEE, 2017.

[54] H. Li, T. F. Wu, A. Rahimi, K.-S. Li, M. Rusch, C.-H. Lin, J.-L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, W.-C. Chiu, M.-C. Chen, T.-T. Wu, J.-M. Shieh, W.-K. Yeh, J. M. Rabaey, S. Mitra, and H.-S. P. Wong, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 16.1.1–16.1.4.

[55] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[56] A. Gebregiorgis, S. Kiamehr, F. Oboril, R. Bishnoi, and M. B. Tahoori, "A cross-layer analysis of soft error, aging and process variation in near threshold computing," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 205–210.

[57] E. A. Antonelo and B. Schrauwen, "On learning navigation behaviors for small mobile robots with reservoir computing architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, pp. 763–780, 2015.

[58] A. Jalalvand, G. V. Wallendael, and R. Walle, "Real-time reservoir computing network-based systems for detection tasks on visual contents," *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, pp. 146–151, 2015.

[59] H. Jaeger, “Adaptive nonlinear system identification with echo state networks,” in *NIPS*, 2002.

[60] T. Natschläger, W. Maass, and H. Markram, “The "liquid computer": A novel strategy for real-time computing on time series,” 2002.

[61] Y. Kume, S. Bian, and T. Sato, “A tuning-free hardware reservoir based on mosfet crossbar array for practical echo state network implementation,” *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 458–463, 2020.

[62] S. Nichele and M. S. Gundersen, “Reservoir computing using non-uniform binary cellular automata,” 2017.

[63] M. Cook, “Universality in elementary cellular automata,” *Complex Systems*, vol. 15, 01 2004.

[64] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[65] A. Rahimi, T. F. Wu, H. Li, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, “Hyperdimensional computing nanosystem,” *ArXiv*, vol. abs/1811.09557, 2018.

[66] S. Aygun, M. S. Moghadam, M. H. Najafi, and M. Imani, “Learning from hypervectors: A survey on hypervector encoding,” *CoRR*, vol. abs/2308.00685, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.00685

[67] M. Imani, D. Kong, A. Rahimi, and T. Rosing, “Voicehd: Hyperdimensional computing for efficient speech recognition,” in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, 2017, pp. 1–8.

[68] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, “Dual: Acceleration of clustering algorithms using digital-based processing in-memory,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 356–371.

[69] D. Dua and C. Graff, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/ISOLET

[70] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, “Human activity recognition on smartphones using a multiclass hardware-friendly support vector

machine," in *AAL*, J. Bravo, R. Hervás, and M. Rodríguez, Eds. Springer, 2012, pp. 216–223.

[71] http://mplab.ucsd.edu, "The MPLab GENKI Database."

[72] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[73] P. Poduval, Z. Zou, H. Najafi, H. Homayoun, and M. Imani, "Stochd: Stochastic hyperdimensional system for efficient and robust learning from raw data," in *IEEE/ACM DAC*, 2021.

[74] P. Poduval, Y. Ni, Y. Kim, K. Ni, R. Kumar, R. Cammarota, and M. Imani, "Hyperdimensional self-learning systems robust to technology noise and bit-flip attacks," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

[75] S. Zhang, R. Wang, D. Ma, J. J. Zhang, X. Yin, and X. Jiao, "Energy-efficient brain-inspired hyperdimensional computing using voltage scaling," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 52–55.

[76] C. Taiyu, "Energy-efficient dnn training with approximate computing and voltage scaling," PhD thesis, Osaka University, Osaka, January 2021, available at https://ir.library.osaka-u.ac.jp/repo/ouka/all/82288/31963_Dissertation.pdf.

[77] T. Song, W. Rim, S. Park, Y. Kim, G. Yang, H. Kim, S. Baek, J. Jung, B. Kwon, S. Cho, H. Jung, Y. Choo, and J. Choi, "A 10 nm finfet 128 mb sram with assist adjustment system for power, performance, and area optimization," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 240–249, 2017.

[78] I. J. Chang, D. Mohapatra, and K. Roy, "A priority-based 6t/8t hybrid sram architecture for aggressive voltage scaling in video applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 2, pp. 101–112, 2011.

[79] M. Cho, J. Schlessman, W. Wolf, and S. Mukhopadhyay, "Accuracy-aware sram: A reconfigurable low power sram architecture for mobile multimedia applications," in *2009 Asia and South Pacific Design Automation Conference*, 2009, pp. 823–828.

[80] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[81] R. E. Schapire, "Explaining adaboost," in *Empirical inference*. Springer, 2013, pp. 37–52.

[82] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, 1970.

[83] S. Wolfram, *A New Kind of Science*. Wolfram Media, 2002. [Online]. Available: https://www.wolframscience.com

[84] J. Buckman, A. Roy, C. Raffel, and I. J. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," in *ICLR*, 2018.

[85] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, pp. 131–155, 2012.

[86] Wikipedia contributors, "Murmurhash — Wikipedia, the free encyclopedia," 2021, [Online; accessed 13-January-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MurmurHash&oldid=1062710708

[87] I. Aleksander, M. Gregorio, F. França, P. Lima, and H. Morton, "A brief introduction to weightless neural systems," in *ESANN*, 2009.

[88] Z.-H. Zhou, *Ensemble Learning*. Boston, MA: Springer US, 2009, pp. 270–273.

[89] R. Polikar, "Ensemble learning," *Scholarpedia*, vol. 4, no. 1, p. 2776, 2009, revision #186077.

[90] D. Liang, M. Hashimoto, and H. Awano, "Bloomca: A memory efficient reservoir computing hardware implementation using cellular automata and ensemble bloom filter," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 587–590.

[91] D. Liang, J. Shiomi, N. Miura, M. Hashimoto, and H. Awano, "A hardware efficient reservoir computing system using cellular automata and ensemble bloom filter," *IEICE TRANSACTIONS on Information and Systems*, vol. 105, no. 7, pp. 1273–1282, 2022.

[92] C.-Y. Chang, Y.-C. Chuang, C.-T. Huang, and A.-Y. Wu, "Recent progress and development of hyperdimensional computing (hdc) for edge intelligence," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2023.

[93] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, 06 2009.

[94] H. Amrouch, M. Imani, X. Jiao, Y. Aloimonos, C. Fermuller, D. Yuan, D. Ma, H. E. Barkam, P. R. Genssler, and P. Sutor, "Brain-inspired hyperdimensional computing for ultra-efficient edge ai," in *2022 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2022, pp. 25–34.

[95] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–40, 2022.

[96] D. Liang, J. Shiomi, N. Miura, and H. Awano, "Distrihd: A memory efficient distributed binary hyperdimensional computing architecture for image classification," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 43–49.

[97] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *ArXiv*, vol. abs/1812.01718, 2018.

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

[99] R. Balasubramonian, A. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization*, vol. 14, pp. 1–25, 06 2017.

[100] D. Liang, J. Shiomi, N. Miura, and H. Awano, "Stridehd: A binary hyperdimensional computing system utilizing window striding for image classification," *IEEE Open Journal of Circuits and Systems*, pp. 1–1, 2024.

[101] B. Zhai, L. Nazhandali, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, D. Blaauw, and T. Austin, "A 2.60pj/inst subthreshold sensor processor for optimal energy efficiency," in *2006 Symposium on VLSI Circuits, 2006. Digest of Technical Papers.*, 2006, pp. 154–155.

[102] A. Wang and A. Chandrakasan, "A 180-mV Subthreshold FFT Processor using a Minimum Energy Design Methodology," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 1, pp. 310–319, Jan. 2005.

[103] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1211–1220.

[104] Z. Zou, Y. Kim, F. Imani, H. Alimohamadi, R. Cammarota, and M. Imani, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3480958

[105] D. Liang, H. Awano, N. Miura, and J. Shiomi, "Dependablehd: A hyperdimensional learning framework for edge-oriented voltage-scaled circuits," in *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023, pp. 416–422.

[106] I. J. Chang, K. Kang, S. Mukhopadhyay, C. Kim, and K. Roy, "Fast and accurate estimation of nano-scaled sram read failure probability using critical point sampling," in *Proceedings of the IEEE 2005 Custom Integrated Circuits Conference, 2005.*, 2005, pp. 439–442.

[107] K. Takeda, H. Ikeda, Y. Hagihara, M. Nomura, and H. Kobatake, "Redefinition of write margin for next-generation sram and write-margin monitoring circuit," in *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*, 2006, pp. 2602–2611.

[108] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Statistical design and optimization of sram cell for yield enhancement," in *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, 2004, pp. 10–13.

[109] I. J. Chang, D. Mohapatra, and K. Roy, "A priority-based 6t/8t hybrid sram architecture for aggressive voltage scaling in video applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 2, pp. 101–112, 2011.

[110] M. Qin, T. Liu, B. Hou, Y. Gao, Y. Yao, and H. Sun, "A low-latency rdp-cordic algorithm for real-time signal processing of edge computing devices in smart grid cyber-physical systems," *Sensors*, vol. 22, no. 19, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/19/7489

[111] T. Vladimirova and H. Tiggeler, "Fpga implementation of sine and cosine generators using the cordic algorithm," in *Proc. of Military and Aerospace Application of Programmable Devices and Technologies Conference (MAPLD 99)*, 1999, pp. 28–30.

[112] E. Alpaydin and C. Kaynak, "Optical Recognition of Handwritten Digits," UCI Machine Learning Repository, 1998, DOI: https://doi.org/10.24432/C50P49.

[113] D. Liang, H. Awano, N. Miura, and J. Shiomi, "A robust and energy efficient hyperdimensional computing system for voltage-scaled circuits," *ACM Trans. Embed. Comput. Syst.*, sep 2023, just Accepted. [Online]. Available: https://doi.org/10.1145/3620671

[114] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[115] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[116] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020. [Online]. Available: https://arxiv.org/abs/1905.11946

[117] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[118] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," 2019. [Online]. Available: https://arxiv.org/abs/1807.11626

[119] J.-S. Park, J.-W. Jang, H. Lee, D. Lee, S. Lee, H. Jung, S. Lee, S. Kwon, K. Jeong, J.-H. Song, S. Lim, and I. Kang, "9.5 a 6k-mac feature-map-sparsity-aware neural processing unit in 5nm flagship mobile soc," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 152–154.

[120] K. Nose, T. Fujii, K. Togawa, S. Okumura, K. Mikami, D. Hayashi, T. Tanaka, and T. Toi, "20.3 a 23.9tops/w @ 0.8v, 130tops ai accelerator with 16× performance-accelerable pruning in 14nm heterogeneous embedded mpu for real-time robot applications," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 364–366.

[121] Y. Wang, X. Yang, Y. Qin, Z. Zhao, R. Guo, Z. Yue, H. Han, S. Wei, Y. Hu, and S. Yin, "34.1 a 28nm 83.23tflops/w posit-based compute-in-memory macro for high-accuracy ai applications," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 566–568.

[122] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, "7.1 an 11.5tops/w 1024-mac butterfly structure dual-core sparsity-aware neural

processing unit in 8nm flagship mobile soc," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2019, pp. 130–132.

[123] Y. Ju, G. Xu, and J. Gu, "20.4 a 28nm physics computing unit supporting emerging physics-informed neural network and finite element method for real-time scientific computing on edge devices," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 366–368.

[124] H. Mo, W. Zhu, W. Hu, G. Wang, Q. Li, A. Li, S. Yin, S. Wei, and L. Liu, "9.2 a 28nm 12.1tops/w dual-mode cnn processor using effective-weight-based convolution and error-compensation-based prediction," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 146–148.

[125] T.-H. Wen, H.-H. Hsu, W.-S. Khwa, W.-H. Huang, Z.-E. Ke, Y.-H. Chin, H.-J. Wen, Y.-C. Chang, W.-T. Hsu, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, S.-H. Teng, C.-C. Chou, Y.-D. Chih, T.-Y. J. Chang, and M.-F. Chang, "34.8 a 22nm 16mb floating-point reram compute-in-memory macro with 31.2tflops/w for ai edge devices," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 580–582.