DOI: 30.0485/ijearst.10.03.3493 https://ijearst.co.in/

ENGINEERING IN ADVANCED RESEARCH SCIENCE AND TECHNOLOGY

ISSN 2386-7876
Vol.02, Issue.03
June-2026
Pages: 1175-1190

Power-Aware ReRAM-Based Processing-in-Memory Architecture for Hyperdimensional Computing Using Clock Gating

[1]SHEIK RIZWANA, [2]GUTTULA VENKATARAMANA
[1]M. Tech, Dept. of E.C.E, V S M College of Engineering, Ramachandrapuram, AP, India
[2]Assistant Professor, Dept. of E.C.E, V S M College of Engineering, Ramachandrapuram, AP, India

ABSTRACT: Hyperdimensional Computing (HDC) is an emerging computing paradigm inspired by the human brain, where information is represented using high-dimensional vectors known as hyper vectors. HDC offers high parallelism, robustness, fault tolerance, and low computational complexity, making it suitable for next-generation artificial intelligence and edge computing applications. This project presents an efficient architecture for Hyperdimensional Computing with an enhanced memory subsystem using the clock gating technique. In conventional HDC architectures, memory units consume significant dynamic power due to continuous clock switching even during idle conditions. To overcome this limitation, clock gating is incorporated to disable unnecessary clock activity in inactive memory blocks, thereby reducing switching power consumption and improving overall energy efficiency. The proposed architecture consists of encoding, training, and associative memory modules integrated with gated clock-controlled memory units. The design is implemented using hardware description language and analyzed for performance, area, and power optimization. Experimental analysis shows that the use of clock gating significantly reduces dynamic power consumption while maintaining computational accuracy and throughput. The modified HDC architecture is highly suitable for low-power embedded systems, IoT devices, wearable electronics, biomedical systems, and real-time intelligent applications where energy-efficient processing is essential.

Keywords: ReRAM (Resistive Random Access Memory), Processing-in-Memory (PIM), Hyperdimensional Computing (HDC), Clock Gating, Low-Power VLSI Design, Energy-Efficient Architecture, In-Memory Computing, Non-Volatile Memory, Hardware Accelerator, Power Optimization, Memory-Centric Computing

INTRODUCTION: Machine learning has achieved the state-of-the-art performance across a wide range of applications, such as image classification [1], video recognition [2], natural language processing (NLP) [3], and gaming strategies [4], to name a few. Further, deep neural networks (DNNs) can even outperform human-level performance in a few tasks, for example, ImageNet classification [5], and board game Go [4]. Meanwhile, the neural networks complexity and parameter size have skyrocketed over the past few years. Despite the fast advance in general-purpose graphics processing unit (GPGPU), its energy efficiency is still far less than the ultimate

'Intelligence', human brain, which contains 1010 neurons and 1014 synapses but only consumes ~20 Watts [6]. One of the bottlenecks comes from the fact that the von Neumann architecture separates the memory and processing unit, introducing signific To address this, emerging non-volatile memory, especially resistive random-access memory (ReRAM) based processingin-memory (PIM) architecture has been studied to accelerate DNN computation [8-10]. Thanks to the PIM architecture and massive data-level parallelism, significant improvements of computing speed and energy efficiency have been demonstrated [8-11]. Most of the prior ReRAM accelerators focus on accelerating convolutional neural networks (CNNs) but lack supports for recurrent neural networks (RNNs). The core concept of RNN is to utilize the sequential information with a feedback loop, i.e. the output of RNN depends not only on current input but also the previous computation results. In practice, long short-term memory (LSTM) [12] and gated recurrent unit (GRU) [13] are the most popular RNN configurations and have demonstrated unprecedented performance for tasks including Human activity recognition (HAR) [14], video recognition [2], and NLP [3, 15], to name a few. There are two reasons making existing ReRAM CNN accelerators not suitable for RNN computing. First, in CNN, the inputs are independent from each other and the output is computed based on current input. However, this is not true for RNN since the computation of RNN requires both current input and results from previous steps. The time-dependence makes the pipeline in prior ReRAM CNN accelerator designs not feasible for RNN computing. Second, the operations of RNN is different with CNN especially for LSTM and GRU which contain not only matrix-vector multiplication and non-linear functions but also element-wise multiplication. In recent years, the interest in machine learning and especially neural-network based techniques has grown significantly. Since conventional processing units such as CPUs and GPUs do not match the characteristics of machine learning (ML) algorithms, researchers have proposed novel hardware architectures for accelerating these algorithms [1]. On the processor front, the von-Neumann style compute-centric architectures are become increasingly constrained by data movement energy and memory bandwidth, since the data movement between the core and off-chip memory incurs ∼100× higher energy than a floating-point operation [2,3]. To address this issue, researchers have proposed "processing in/near memory" (also called near-data processing) whereby the computation logic is placed inside memory or the characteristic of memory itself is exploited for performing computations [4,5]. This approach avoids data movement completely and, thus, promises to break the memory wall. For design of ML accelerators and processing-in-memory (PIM) solutions, emerging memories, such as resistive RAM (ReRAM) offer distinct advantages over conventional CMOS (complementary metal–oxide–semiconductor) based designs [6,7]. In the CMOS-based approach, modeling a neuron requires tens of transistors. In addition, SRAM (static random access memory) is a volatile memory with high leakage energy [8] and, since SRAM does not efficiently support a wide range of operations in memory, SRAM-based designs such as TrueNorth [1] use separate logic for performing computations. By comparison, ReRAM is a non-volatile memory with near-zero leakage energy and high density. The ReRAM state reflects the current passed through it in the history and this is very useful for modeling the synaptic weights of neurological synapses and implementing neural network (NN) architectures [9–11].

LITERATURE SURVEY: For design of ML accelerators and processing-in-memory (PIM) solutions, emerging memories, such as resistive RAM (ReRAM) offer distinct advantages over conventional CMOS (complementary

metal–oxide–semiconductor) based designs [6,7]. In the CMOS-based approach, modeling a neuron requires tens of transistors. In addition, SRAM (static random access memory) is a volatile memory with high leakage energy [8] and, since SRAM does not efficiently support a wide range of operations in memory, SRAM-based designs such as TrueNorth [1] use separate logic for performing computations. By comparison, ReRAM is a non-volatile memory with near-zero leakage energy and high density. The ReRAM state reflects the current passed through it in the history and this is very useful for modeling the synaptic weights of neurological synapses and implementing neural network (NN) architectures [9–11]. In addition, ReRAM supports operations such as analog matrix-vector multiplication (MVM), search and bitwise operations within memory which facilities energy-efficient accelerator design. For example, since the CONV (convolution) operation in convolutional neural networks (CNNs) involves MVM, and CONV layers account for more than 95% of computations in CNNs, ReRAM-based processing engine can boost the efficiency of CNNs significantly [12,13]. These factors have motivated researchers to implement a variety of ML/NN architectures on ReRAM, such as multi-layer perceptron [14,15], CNN [16–18], tensorized NN [19] and auto-associative memory [14]. Use of ReRAM for designing neuromorphic computing systems (NCSs), however, also presents challenges. For example, ReRAM has several reliability issues, such as limited write endurance, resistance drift, susceptibility to process variation (PV), etc. [20–23]. Analog operation exacerbates these challenges and also brings area/energy overheads of ADCs/DACs (analog-to-digital converters/digital-to-analog converters). Further, precise tuning of ReRAM requires frequent update of weights and large number of training iterations which incurs high overhead due to the high write energy and latency of ReRAM [24,25]. This presents challenges in achieving high throughput and accuracy. Addressing these challenges using (micro)architectural and system-level techniques is vital for ensuring adoption of ReRAM in state-of-the-art neuromorphic computing systems (NCSs). Several recent techniques seek to fulfill this need.

Figure 2 shows the use of memristor for performing dot-product computation. Each bitline connects to each wordline through a ReRAM cell. Let R and G denote the resistance and conductance of a cell, where G = 1/R. If the cells in a column are programmed such that their conductance values are G1, G2, . . . Gk . On applying the voltages V1, V2, . . . Vk to these k rows, a current of Vi × Gi current passes from the cell into bitline, as per Ohm's law. Then, from Kirchoff's law, the total current from the bitline is the sum of currents flowing through each column, as shown in Figure 2a. The total current (I) is the dot-product of input voltages at each row (V) and cell conductances (G) in a column, that is, I = V × G.

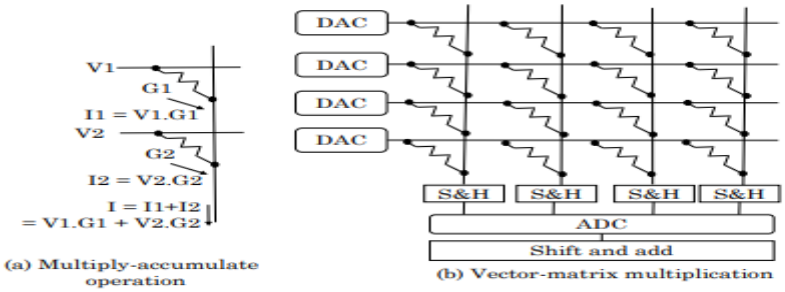

Figure 1. (a) Performing an analog sum-of-products operation using a bitline; (b) using an MCA for MVM

We first summarize CNNs targeted at image detection and classification, such as the winners of the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [58]. These algorithms are judged by their top-5 error rates, i.e., how often is the target label not among the top 5 predictions made by the network. The winners in 2012 (AlexNet [4]), 2013 (Clarifai [80]), and 2014 (GoogLeNet [69]) achieved top5 error rates of 15.3%, 11.2%, and 6.67% respectively. Based on the observations of Jarrett et al. [28], all of these networks incorporate LRN layers.

EXISTING METHOD:

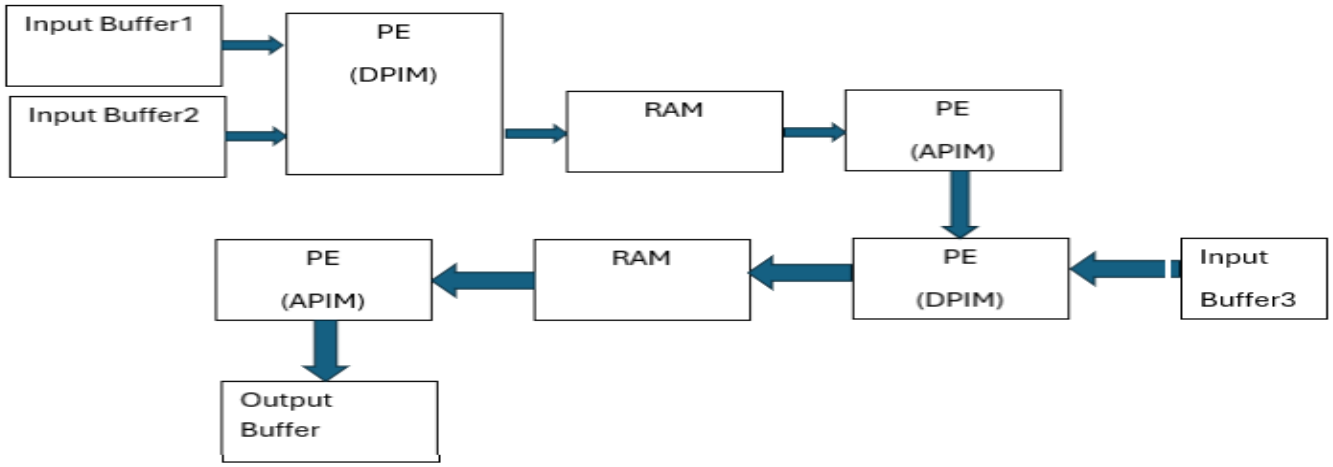

Fig:2 Existing Hyperdimensional Computing block diagram

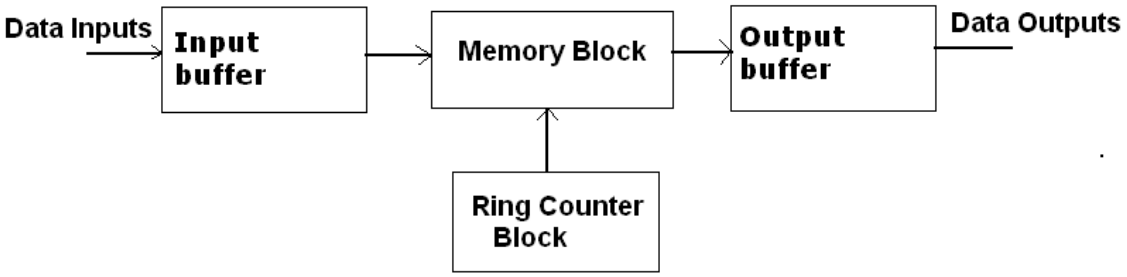

Fig : 3 Existing Block Of Memory Organisation

MEMORY BLOCK: (RAM) Random-access memory (RAM) is a form of computer data storage. Today, it takes the form of integrated circuits that allow stored data to be accessed in any order (that is, at random). "Random" refers to the idea that any piece of data can be returned in a constant time, regardless of its physical location and whether it is related to the previous piece of data. The word "RAM" is often associated with volatile types of memory (such as DRAMmemory modules), where the information is lost after the power is switched off. Many other types of memory are RAM as well, including most types of ROM and a type of flash memory called NOR-Flash. Scan design has been the backbone of design for testability (DFT) in industry for about three decades because scan-based design can successfully obtain controllability and observability for flip-flops. Serial Scan design has dominated the test architecture because it is convenient to build. However, the serial scan design causes unnecessary switching activity during testing which induce unnecessarily enormous power dissipation. The test time also increases dramatically with the continuously increasing number of flip-flops in large sequential circuits even using multiple scan chain

architecture. An alternate to serial scan architecture is Random Access Scan (RAS). In RAS, flip-flops work as addressable memory elements in the test mode which is a similar fashion as random access memory (RAM). This approach reduces the time of setting and observing the flip-flop states but requires a large overhead both in gates and test pins. Despite of these drawbacks, the RAS was paid attention by many researchers in these years. This paper takes a view of recently published papers on RAS and rejuvenates the random access scan as a DFT method that simultaneously address three limitations of the traditional serial scan namely, test data volume, test application time, and test power.

RING COUNTER: A **ring counter** is a type of counter composed of a circular shift register. The output of the last shift register is fed to the input of the first register. There are two types of ring counters: A *straight ring counter* or *Overbeck counter* connects the output of the last shift register to the first shift register input and circulates a single one (or zero) bit around the ring. For example, in a 4-register one-hot counter, with initial register values of 1000, the repeating pattern is: 1000, 0100, 0010, 0001, 1000... . Note that one of the registers must be pre-loaded with a 1 (or 0) in order to operate properly. A *twisted ring counter* (also called *Johnson counter* or *Moebius counter*) connects the complement of the output of the last shift register to its input and circulates a stream of ones followed by zeros around the ring. For example, in a 4-register counter, with initial register values of 0000, the repeating pattern is: 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000. If the output of a shift register is fed back to the input. aringcounter results. The data pattern contained within the shift register will recirculate as long as clock pulses are applied.

PROPOSED METHOD:

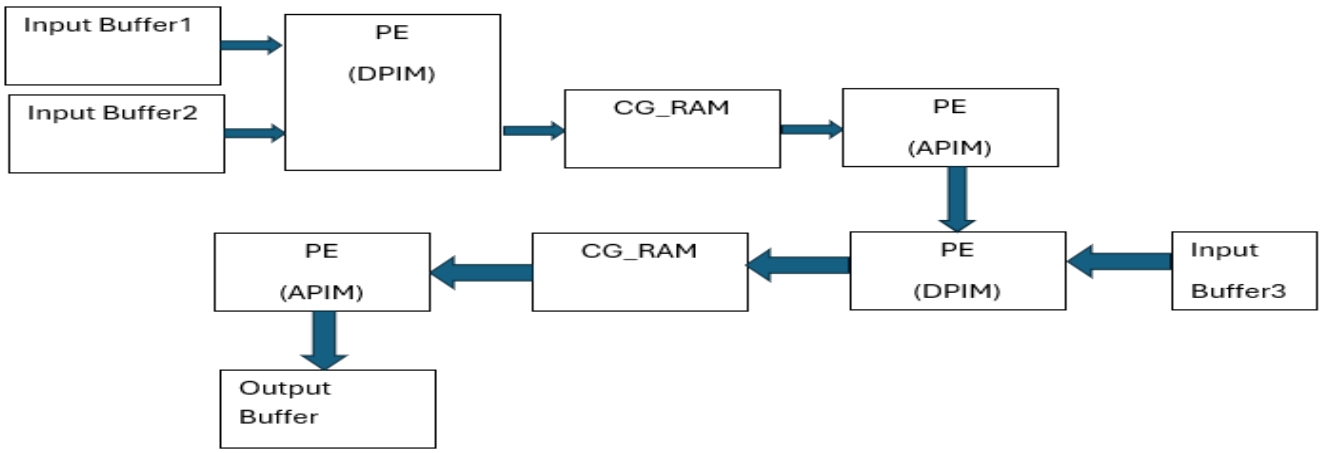

Fig:4 Proposed Hyperdimensional Computing block diagram

HDC is a novel computing model inspired by the way that neurons work in human brains [19]. It has been applied in various areas, such as image recognition [4] and text classification [20]. The basic data structure in HDC is the high-dimensional vector (i.e., HV) whose dimension may exceed 10 000. HVs can represent different features of real items in various real-world recognition tasks, such as language characters and bio-signals. An HV is generated randomly and holographically with independent and identically distributed (i.i.d.) elements which are binary (0/1) or bipolar (+1/−1) codes [21]. For all elements in an HV, no one stores more information than the others, thus improving the

robustness of HDC. When the dimension of HVs is rather high (e.g., D = 10 000), any two random HVs are almost orthogonal. There are several typical operations in HDC [22], such as binding operations that bind two HVs to generate a new HV, and bundling operations that bundle many HVs together into a new HV. The binding operation retains one half of the information from input HVs, while the bundling operation generates new HVs that are orthogonal to the input HVs [23]. These operations sustain the dimension of HVs, and thus can keep the computing paradigm of HDC stable. In the following, we describe the two key phases of HDC inference. 1) Encoding Phase: As shown in Fig. 1, the first step of HDC is to map the real-world data into a hyperdimensional space to generate HVs. This operation is called the encoding phase. Several encoding approaches have been proposed in previous studies [5], [24]. Generally, they linearly sum up all HVs associated with each feature. A feature is composed of a position HV and a level HV. Position and level HVs are

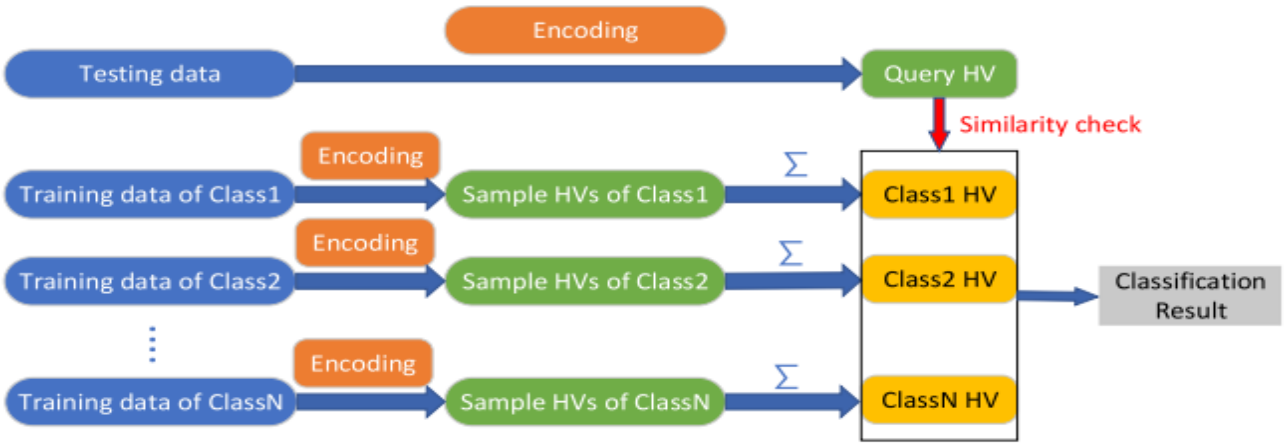

Fig. 5. HDC process for classification tasks.

generated in the same way for both training and inference. In the encoding phase, an n-dimensional feature vector is converted into nD-dimensional HVs, where D represents the number of dimensions and each element in the HV is a binary number. The encoder looks for the minimum and maximum feature values and linearly quantizes the range into m levels. At the same time, it generates a D-dimensional random binary HV for each level, i.e., {L1,..., Lm}. Similarly, n feature indexes are encoded into n random D-dimensional binary HVs, i.e., {P1,..., Pn}. These position HVs are generated randomly and any two position HVs are orthogonal to each other. In contrast, there are correlations between level HVs. For example, if two feature values are closer, the similarity between their level HVs is higher. Assume $\vec{P_m}$ and $\vec{L_m}$ represent the position HV and the level HV of the mth feature, respectively, the mth feature HV can be defined as follow:

$$\vec{F_m} = \vec{P_m} \oplus \vec{L_m}$$

where $\oplus$ is an bitwise binding (i.e., XOR) operation between two HVs. Assume there are total n features in a sample, HDC combines all feature HVs to generate the sample HV, as shown in

$$\vec{H} = \sum_{m=1}^{n} \vec{F}_m.$$

2) Comparison Phase: During the HDC inference, the queried data are converted into a query HV through the encoding phase. Then, the similarity between the query HV and all class HVs are calculated based on the Hamming distance. The queried data are classified into a class when their Hamming distance is the smallest. At last, the class label with the highest similarity is returned to the query request. The HDC training includes an encoding phase and an accumulation phase. Each training sample is encoded into a sample HV in the encoding phase. Then, all sample HVs that belong to the same class are accumulated to characterize general attributes of the class in the training dataset. Assuming there are total s samples in the lth class, the lth class HV can be represented by (3). In the end, all class HVs are stored in the associative memory for comparison

$$\vec{C}_l = \sum_{i=1}^{s} \vec{H}_i.$$

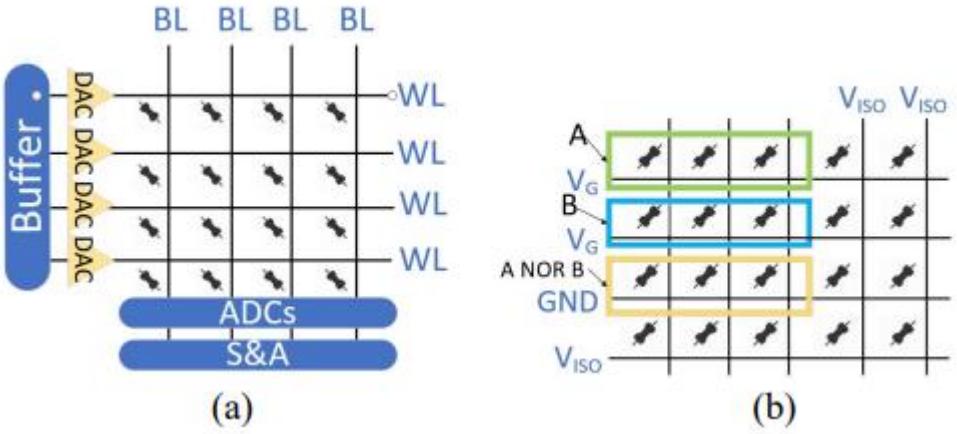

Fig. 6. Two kinds of PIM architecture. (a) ReRAM-based APIM. (b) ReRAM-based DPIM.

B. ReRAM-Based PIM Accelerators ReRAM is a typical nonvolatile memory (NVM) that operates by changing the resistance state of ReRAM cells [25]. The metal-insulator-metal (MIM) structure of an ReRAM cell comprises an upper electrode, a lower electrode, and a metal oxide layer sandwiched between them. By applying an external voltage, an ReRAM cell can achieve state transitions between a high resistance state (HRS) and a low resistance state (LRS) which correspond to logic “0” and “1,” respectively. A set operation changes HRS (0) to LRS (1). To set an ReRAM cell, a positive voltage should be applied to generate sufficient write current. Since an ReRAM cell can sustain up to 1012 times [15] writes, its endurance issue is less significant compared to other NVMs. ReRAM-based PIM accelerators enable in-situ computing which eliminates extensive data movement between processors and main memory, and thus achieve significant reduction of energy consumption and execution time [26]. Therefore, ReRAM-based PIM accelerators are extensively explored to enhance the performance of data-centric applications, including

DNNs [13], graph processing [15], and database operations [27]. Generally, PIM architectures can be categorized into APIM and DPIM based on their analog or digital computing paradigms, respectively. 1) ReRAM-Based APIM Architecture: Generally, most APIM accelerators are designed to accelerate the MVM operations [28]. As shown in Fig. 2(a), APIM performs MVM operations through the ReRAM crossbar array. At first, digital signals are transformed into analog signals via digital-toanalog converters (DACs). After applying a voltage to the wordline of ReRAM cells, a corresponding current is generated on the bitline based on Ohm's law. The currents on the same bitline are then accumulated using Kirchhoff's law. This enables APIM to perform array-level MVM operations in a single cycle. The analog signal is subsequently transferred for further calculations. Similarly, in the output circuit, analog-to-digital converters (ADCs) are employed to convert output analog results into digital values. This fixed computing paradigm of APIM is only applicable to a few applications that contain a large proportion of MVM operations.

2) ReRAM-Based DPIM Architecture: A DPIM is able to perform in-situ NOR operations without additional peripheral circuits [17]. Fig. 2(b) demonstrates the design of the DPIM array. For an ReRAM cell, there is a significant difference

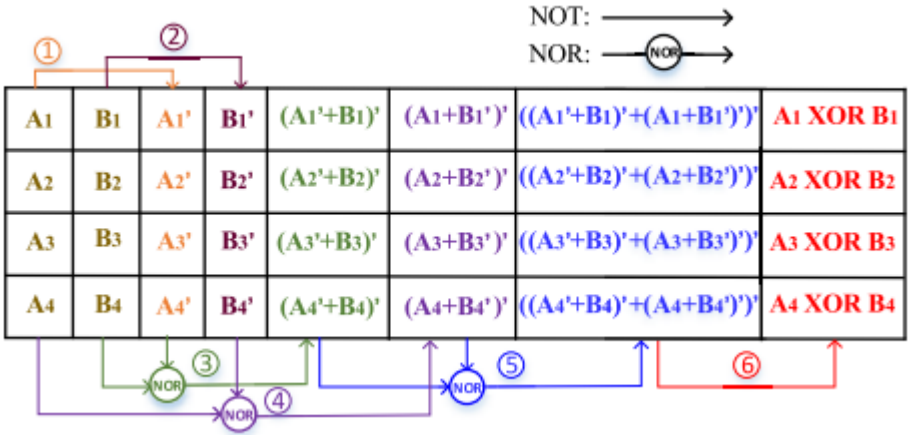

Fig. 7. Processing of a XOR operation in a DPIM.

between the HRS and the LRS. The initial output ReRAM cell should be set to the LRS. According to Kirchhoff's law, in a parallel circuit, the overall resistance of input ReRAM cells is lower than that of each individual input ReRAM cell. Thus, when an input ReRAM cell is in an LRS, the current flowing out the circuit increases, allowing the output ReRAM cell to change from an LRS to an HRS. The circuit characteristic described above can be exploited to implement a logical NOR operation (i.e., $A \oplus B = (A + B)$), which can be represented as follows: 0 NOR 0 = 1, 1 NOR 1 = 0, and 0 NOR 1 = 0. Thus, the DPIM naturally supports NOR operations. Furthermore, the DPIM array can also support row or column-level parallelism, and thus has higher elasticity than the APIM. As most arithmetic operations can be converted into a series of NOR operations, the DPIM is able to flexibly execute arithmetic and logical operations. For example, the 1-bit XOR logic operation can be implemented with 6 NOR operations based on De Morgan's law. Assume A and B represent the input two operands, and A represents NOT operation on A. The XOR operation $A \oplus B$ can be represented by (4). Fig. 3 illustrates the six steps of the XOR operation in a DPIM

$$
\begin{aligned}
A \oplus B &= A \cdot B' + A' \cdot B \\
&= (A' + B)' + (A + B')' \\
&= \Big(\big((A' + B)' + (A + B')' \big)' \Big)'.
\end{aligned}
$$

The Limitation of Computing Paradigms: Traditional APIM accelerators are designed for accelerating MVM operations. However, in HDC applications, other arithmetic and logical operations like bitwise XOR cannot be efficiently executed by APIM, posing a significant limitation for fully utilizing ReRAM-based HDC accelerators. In contrast, DPIM accelerators can perform NOR operations in parallel. Nevertheless, the DPIM is much slower than APIM when it performs MVM operations in HDC. Hence, the integration of APIM and DPIM is crucial for effectively accelerating HDC applications which are comprised of both MVM and bitwise logical operations. Disparities of ReRAM Resource Requirements Between the Encoding and Comparison Phases for Various Datasets: The

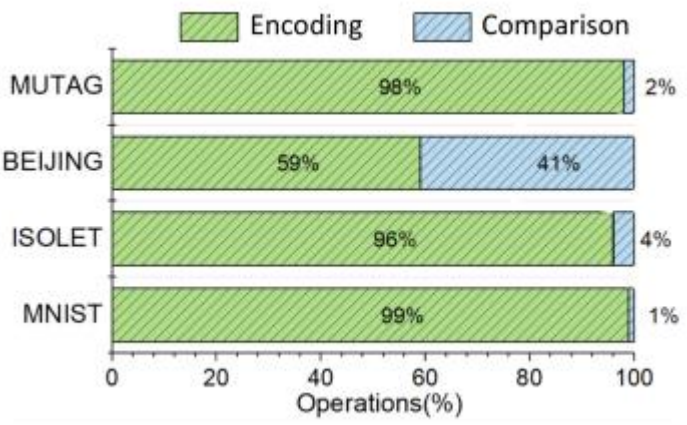

Fig. 8. Breakdown of operations in encoding and comparison phases for different datasets.

most recent ReRAM-based HDC accelerator [4] allocates fixsized ReRAM resource to processing units for the encoding and comparison phases, and thus cannot fairly satisfy the resource requirement of these two phases for different datasets. Fig. 4 shows the breakdown of operations in the encoding and comparison phases of HDC for four datasets. We can find that the percentage of operations used for the encoding phase and the comparison phase varies significantly for different datasets. For instance, the comparison phase accounts for only 1% total operations in the MNIST dataset, while there are 41% comparison operations in the BEIJING dataset [29]. The proportion of operations in the encoding and comparison phases is mainly determined by the number of features and classes in the dataset. A dedicated hardware architecture designed for these two phases cannot adapt to different datasets, usually resulting in low utilization of computing resource. This motivates us to develop a unified ReRAM-based HDC accelerator that can dynamically allocate PEs to the two phases based on their ratio of operations. In this way, our

architecture can adapt to various datasets more flexibly, and improves the utilization of ReRAM resource and the performance of HDC applications. Fixed Access Paths of APIM Arrays: APIM plays a crucial role in ReRAM-based HDC accelerators. During the encoding phase, APIM accumulates feature HVs to obtain the sample HV. During the comparison phase, APIM calculates the Hamming distance between two HVs. However, conventional APIM arrays only support bitline accumulation in one direction, and thus limits the utilization of APIM in an ReRAM-based HDC accelerator. To enable these two types of computations for HDC applications, it is essential to redesign the APIM architecture so that it can perform MVM operations in two directions. Write Overhead Caused by Data Dependencies: During the HDC processing, data dependencies exist in both encoding and comparison phases. In addition, while multiple ReRAM arrays offer the potential for parallel computing, the large amount of data involved in HDC processing results in substantial write overhead during intermediate data transfers, leading to performance degradation. Therefore, it is essential to hide the write latency of ReRAM. These challenges motivate us to design an ReRAM-based accelerator for general HDC processing. Particularly, it can be dynamically configured according to the characteristics of different datasets.

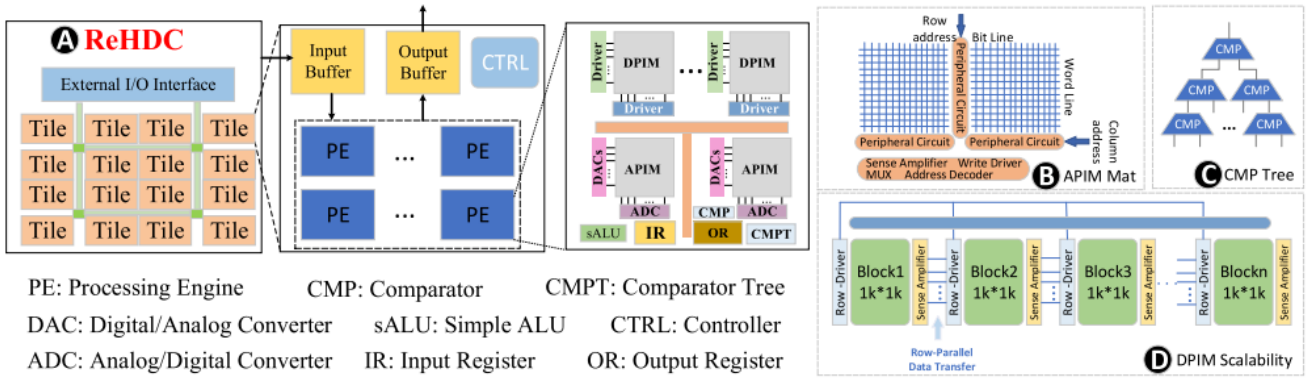

Fig. 9. Overview of ReHDC architecture

Overview of ReHDC

Fig. 5 demonstrates the architecture of ReHDC. As shown in Fig. 5(a), ReHDC is comprised of multiple tiles that are interconnected through a global bus. Each tile is composed of a controller (CTRL), several PEs, and two buffers. The input buffer is responsible for caching data that is written to the DPIM, while the output buffer stores the results of the PE. The controller generates the control signals for each component. sALU is responsible for computing the sum of the bitlines in multiple arrays and performing other simple operations. In each PE, the DPIM crossbar arrays are utilized to perform element-wise XOR logic operations, while the APIM crossbar arrays are employed for array-level accumulation operations. As shown in Fig. 5(b), each wordline in the APIM is connected to a DAC, which converts the input data into a vector of voltages and applies it to each bitline of the crossbar array. In contrast, the ADC converts the output current into digital signals for further processing, and is shared by multiple bitlines of the crossbar array. To capture the intermediate data of PEs, each PE contains both an input register (IR) and an output register (OR) to latch the data. Each PE also contains multiple Comparators (CMPs). As shown in Fig. 5(c), some CMPs are arranged as a CMP tree to determine the minimum Hamming distance between the sample HV and each class HV, and the others are employed to binarize the output results of the APIM in the encoding phase based on a

predetermined threshold. To improve data parallelism for DPIMs, multiple cascaded DPIM arrays are exploited to support row-level parallel data transfer and element-wise XOR operations, as shown in Fig. 5(d). B. Data Processing in the Encoding Phase The encoding phase of HDC mainly consists of two operations: 1) binding and 2) bundling. Binding is employed to handle element-wise XOR operations between the level HV

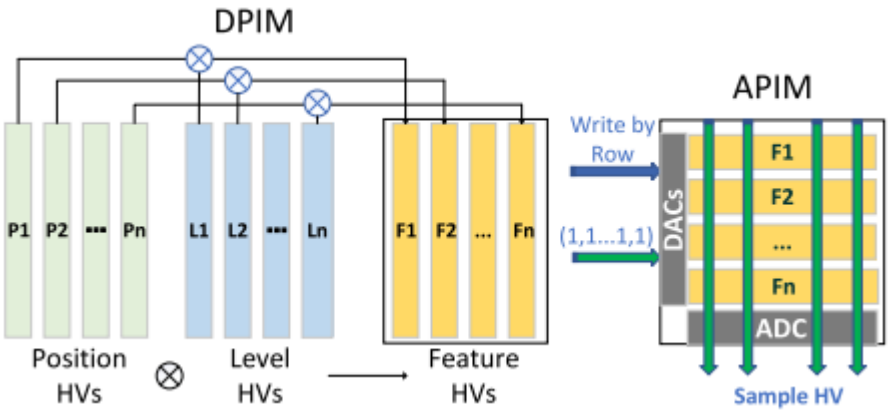

Fig. 10 Data mapping in the encoding phase.

and position HV of each feature. The bundling operation is used to accumulate multiple feature HVs of a sample. We choose DPIM arrays to perform element-wise XOR operations with high parallelism. We employ a judicious HV-to-DPIM mapping scheme to leverage row parallelism and perform element-wise XOR operations. Given that an HV in HDC usually contains thousands of dimensions, element-wise XOR using DPIM can significantly exploit data parallelism and enhance the efficiency of the encoding phase in HDC. To perform multiple XOR operations in parallel, both two of their operands should be placed in the same row of a crossbar, and the output results are stored in the same row of the crossbar as well. Fig. 6 shows the data mapping of the encoding phase. We partition the DPIM array into three regions which are used for storing preprocessed positions HVs, level HVs, and output HVs. The size of crossbar arrays assigned for storing input position HVs and level HVs is identical. We map the preprocessed input HVs in columns of DPIMs and then perform element-wise XOR operations. The generated feature HVs are then stored in the output region. Multiple DPIMs can perform the XOR operation concurrently to maximize data parallelism. In this way, the computation latency can be significantly reduced. When the XOR operation is completed in DPIMs, we use APIMs to perform accumulation operations since APIMs exhibits higher performance than DPIMs for MVM operations.

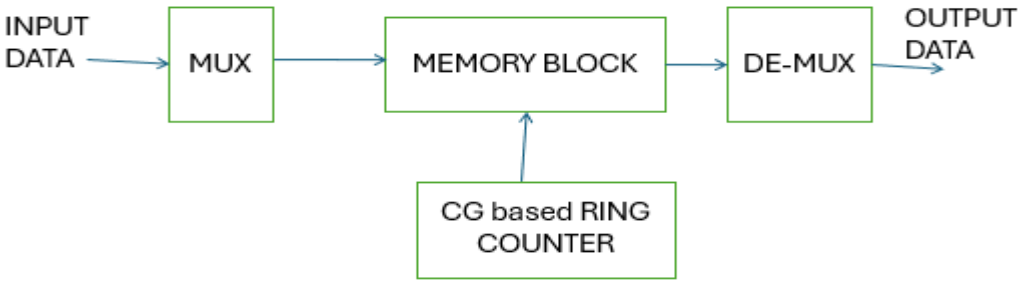

Fig:11 Memory organization using CG

CLOCK GATING: Clock gating is a widely used power-saving technique in digital circuit design (like ASICs and SoCs) that selectively blocks the clock signal from reaching inactive circuit blocks. By preventing flip-flops from unnecessarily toggling when their data isn't changing, it significantly reduces dynamic power consumption—often yielding up to 30% or more in power savings.

The Core Concept: In synchronous systems, the clock tree constantly toggles, consuming power even if the data being processed remains the same. Clock gating inserts combinational logic (such as an AND gate) into the clock path to act as a switch.

Enable Signals: When a specific module is idle, an enable signal becomes 0, stopping the clock signal from propagating to that part of the chip

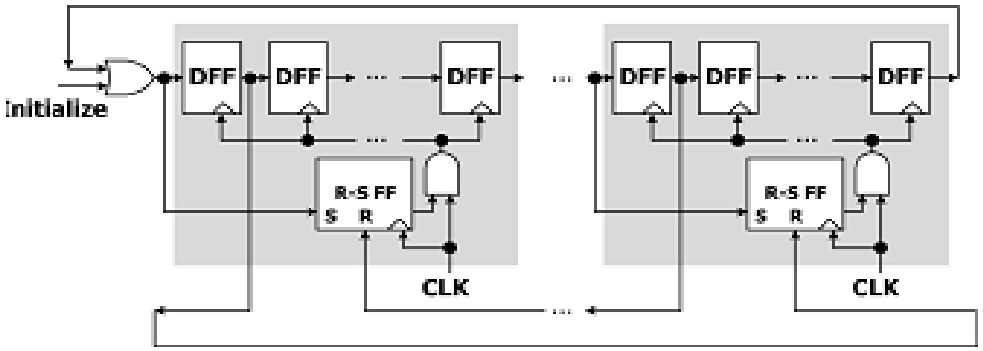

Fig: 12 Ring Counter with SR Flip-Flops

RESULTS:

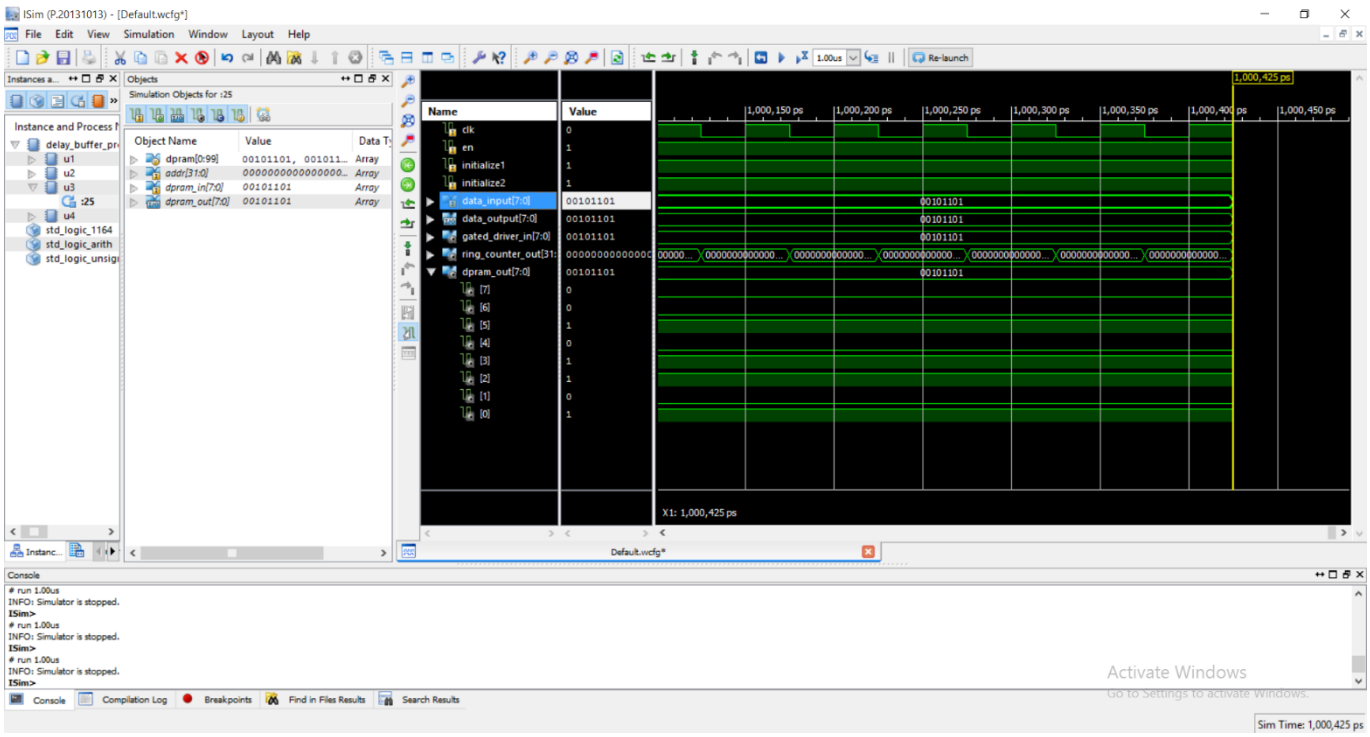

Fig: a Proposed result

Advantages

1. Reduced power consumption due to clock gating in memory blocks.
2. Improved energy efficiency for low-power applications.
3. High fault tolerance and robustness of Hyperdimensional Computing.
4. Faster parallel data processing capability.
5. Lower switching activity in inactive modules.
6. Better scalability for large datasets and AI applications.
7. Suitable for hardware implementation using FPGA or ASIC.
8. Reduced heat generation and improved system reliability.

Applications

- Artificial Intelligence and Machine Learning systems
- Internet of Things (IoT) devices
- Wearable healthcare monitoring systems
- Edge computing architectures
- Biomedical signal processing
- Image and speech recognition systems
- Brain-inspired neuromorphic computing
- Smart sensor networks

Conclusion: The proposed Hyperdimensional Computing architecture with clock gating-based memory optimization provides an effective solution for reducing power consumption in modern intelligent computing systems. By integrating clock gating techniques into the memory subsystem, unnecessary clock transitions are minimized, leading to significant reductions in dynamic power dissipation. The architecture maintains the inherent advantages of HDC such as robustness, parallelism, and fault tolerance while improving overall energy efficiency. The design is highly suitable for low-power and real-time applications including IoT, wearable devices, and AI accelerators. The implementation demonstrates that combining HDC with power optimization techniques can achieve better performance and efficient hardware utilization for future intelligent systems.

Future Scope: Integration with advanced non-volatile memories such as ReRAM and MRAM. Optimization using advanced low-power VLSI techniques. Development of adaptive clock gating mechanisms based on workload conditions. Extension of the architecture for deep learning and neuromorphic computing applications. Integration with edge AI and cloud computing platforms. Enhancement of memory efficiency using compression and sparse encoding techniques. Exploration of security-oriented HDC architectures for encryption and authentication. Performance improvement using pipelined and parallel processing architectures.

REFERENCES:

1. S. Imani, T. F. Wu and T. Rosing, “A Framework for Collaborative Learning in Hyperdimensional Computing,” *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1523–1535, Oct. 2019.

2. A. Rahimi, P. Kanerva and J. M. Rabaey, "Hyperdimensional Computing for Noninvasive Brain–Computer Interfaces: Blind and One-Shot Classification of EEG Error-Related Potentials," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, Jan. 2019.
3. M. Hersche, M. Karunaratne, A. Sebastian and A. Rahimi, "Constrained Few-Shot Class Learning Using Hyperdimensional Computing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 7, pp. 3157–3169, Jul. 2022.
4. D. Kleyko, E. Osipov and A. Senior, "Hyperdimensional Computing: A Framework for Stochastic Computation and Symbolic AI," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 2, pp. 185–198, Apr. 2021.
5. A. Rahimi et al., "Efficient Biosignal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, Jan. 2019.
6. P. Kanerva, "Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, Jun. 2009.
7. M. Karunaratne, M. Hersche, A. Sebastian and A. Rahimi, "Scaling Hyperdimensional Computing Across Modern Hardware," *IEEE Micro*, vol. 40, no. 3, pp. 41–51, May–Jun. 2020.
8. F. Montagna, A. Rahimi and D. Rossi, "PULP-HD: Accelerating Brain-Inspired High-Dimensional Computing on a Parallel Ultra-Low Power Platform," in *Proc. Design, Automation & Test in Europe Conference (DATE)*, Grenoble, France, 2020, pp. 131–136.
9. C. Li et al., "Hyperdimensional Computing with Emerging Memory Technologies," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 271–285, Jun. 2019.
10. S. Datta, A. Pal and A. Basu, "Stochastic Computing and Hyperdimensional Computing for Energy-Efficient AI Hardware," *IEEE Transactions on Circuits and Systems I*, vol. 69, no. 4, pp. 1520–1532, Apr. 2022.
11. M. Imani, S. Salamat, A. Khaleghi and T. Rosing, "VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition," in *Proc. IEEE International Conference on Rebooting Computing (ICRC)*, San Diego, CA, USA, 2017, pp. 1–8.
12. A. Thomas, S. Datta and A. Basu, "A Low-Power VLSI Architecture for Hyperdimensional Computing Using Clock Gating Technique," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Austin, TX, USA, 2022, pp. 1–5.
13. M. Karunaratne et al., "Energy Efficient In-Memory Hyperdimensional Encoding for Spatio-Temporal Signal Processing," *IEEE Transactions on Circuits and Systems I*, vol. 67, no. 12, pp. 4538–4551, Dec. 2020.
14. S. Mittal, "A Survey of ReRAM-Based Architectures for Processing-In-Memory and Neural Networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 75–114, 2019.
15. M. Imani, Y. Kim and T. Rosing, "HDNA: Energy-Efficient DNA Sequencing Using Hyperdimensional Computing," in *Proc. IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Washington, DC, USA, 2018, pp. 271–284.

[16] A. Haj-Ali, R. Ben-Hur, N. Wald, R. Ronen, and S. Kvatinsky, “Not in name alone: A memristive memory processing unit for real in-memory processing,” IEEE Micro, vol. 38, no. 5, pp. 13–21, Oct. 2018.

[17] M. Imani, S. Gupta, Y. Kim, and T. Rosing, “FloatPIM: In-memory acceleration of deep neural network training with high precision,” in Proc. 46th Int. Symp. Comput. Archit., 2019, pp. 802–815. [18] J. Liu, M. Ma, Z. Zhu, Y. Wang, and H. Yang, “HDC-IM: Hyperdimensional computing in-memory architecture based on RRAM,” in Proc. 26th IEEE Int. Conf. Electron., Circuits Syst., 2019, pp. 450–453.

[19] P. Kanerva, “Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors,” Cogn. Comput., vol. 1, no. 2, pp. 139–159, 2009.

[20] P. Alonso, K. Shridhar, D. Kleyko, E. Osipov, and M. Liwicki, “HyperEmbed: Tradeoffs between resources and performance in NLP tasks with hyperdimensional computing enabled embedding of N-gram statistics,” in Proc. Int. Joint Conf. Neural Netw., 2021, pp. 1–9. [21] A. Kazemi, M. M. Sharifi, Z. Zou, M. Niemier, X. Sharon Hu, and M. Imani, “Mimhd: Accurate and efficient hyperdimensional inference using multi-bit in-memory computing,” in Proc. IEEE/ACM Int. Symp. Low Power Electron. Design, 2021, pp. 1–6. [22] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, “In-memory hyperdimensional computing,” Nature Electron., vol. 3, no. 6, pp. 327–337, 2020. [23] S. Zhang, R. Wang, J. J. Zhang, A. Rahimi, and X. Jiao, “Assessing robustness of hyperdimensional computing against errors in associative memory,” in Proc. 32nd Int. Conf. Appl.-Specific Syst., Archit. Process., 2021, pp. 211–217.

[24] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, “Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception,” Sci. Robot., vol. 4, no. 30, 2019, Art. no. eaaw6736. [25] H. Akinaga and H. Shima, “Resistive random access memory (ReRAM) based on metal oxides,” Proc. IEEE, vol. 98, no. 12, pp. 2237–2251, Dec. 2010. [26] H. Li, H. Jin, L. Zheng, Y. Huang, and X. Liao, “ReCSA: A dedicated sort accelerator using ReRAM-based content addressable memory,” Front. Comput. Sci., vol. 17, no. 2, 2023, Art. no. 172103. [27] H. Li, H. Jin, L. Zheng, and X. Liao, “ReSQM: Accelerating database operations using ReRAM-based content addressable memory,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 39, no. 11, pp. 4030–4041, Nov. 2020. [28] P.-Y. Chen, X. Peng, and S. Yu, “NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 12, pp. 3067–3080, Dec. 2018. [29] S. Zhang, B. Guo, A. Dong, J. He, Z. Xu, and S. X. Chen, “Cautionary tales on air-quality improvement in Beijing,” Proc. R. Soc. A, Math., Phys. Eng. Sci., vol. 473, no. 2205, 2017, Art. no. 20170457. [30] S. Li et al., “RC-NVM: Dual-addressing non-volatile memory architecture supporting both row and column memory accesses,” IEEE Trans. Comput., vol. 68, no. 2, pp. 239–254, Feb. 2019. [31] H. Liu, J. Xu, X. Liao, H. Jin, Y. Zhang, and F. Mao, “A simulation framework for memristor-based heterogeneous computing architectures,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 41, no. 12, pp. 5476–5488, Dec. 2022. [32] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, “VTEAM: A general model for voltage-controlled memristors,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 62, no. 8, pp. 786–790, Aug. 2015.