On the Reliability of Computing-in-Memory Accelerators for Deep Neural Networks

Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi

Abstract Computing-in-memory with emerging non-volatile memory (nvCiM) is shown to be a promising candidate for accelerating deep neural networks (DNNs) with high energy efficiency. However, most non-volatile memory (NVM) devices suffer from reliability issues, resulting in a difference between actual data involved in the nvCiM computation and the weight value trained in the data center. Thus, models actually deployed on nvCiM platforms achieve lower accuracy than their counterparts trained on the conventional hardware (e.g., GPUs). In this chapter, we first offer a brief introduction to the opportunities and challenges of nvCiM DNN accelerators and then show the properties of different types of NVM devices. We then introduce the general architecture of nvCiM DNN accelerators. After that, we discuss the source of unreliability and how to efficiently model their impact. Finally, we introduce representative works that mitigate the impact of device variations.

Keywords Compute-in-memory (CIM) · Device variations · Deep neural networks (DNN)

1 Introduction

Deep Neural Networks (DNNs) have excelled human performance in various crucial tasks (e.g., image classification, object detection, and speech recognition) and have become a popular solution for them. Thus, edge devices such as automobiles, smartphones, and smart sensors that depend on these tasks are ideal platforms to be empowered by DNNs. However, due to the constrained computation resource and limited

Z. Yan · X. S. Hu · Y. Shi (✉)
University of Notre Dame, Notre Dame, IN, USA
e-mail: yshi4@nd.edu

Z. Yan
e-mail: zyan2@nd.edu

X. S. Hu
e-mail: shu@nd.edu

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_9

power budget of edge devices, direct implementation of computational intensive DNNs on edge devices is a significant challenge.

A majority of the works addressing this challenge use application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs) for DNN acceleration. These conventional special-purpose edge DNN accelerators typically use a group of on-chip process elements (PEs) to handle computation and utilize off-chip non-volatile (NV) storage (e.g., flash) to store the model information (i.e., DNN architecture and model weights) [8]. Between the static random-access memory (SRAM) in the PEs used for temporary data caching and off-chip non-volatile storage used for power-off data preservation, there is also a complex memory hierarchy, generally consisting of several levels of dynamic random access memory (DRAM)-based on-chip memories. Because of this separation of data and computation, which is a key limitation of the conventional von-Neumann architecture, these DNN edge accelerators face energy efficiency and computing latency challenges. Specifically, PEs of this kind of architecture generates a large volume of intermediate data. These intermediate data need to be moved between different levels of the memory hierarchy so that they can be used by different process elements. Data movements across different levels of memory hierarchy induce a great time and energy consumption overhead, especially when accessing the lower level of the memory hierarchy. This challenge is also called *the memory wall*.

Non-volatile Computing-in-Memory (nvCiM) DNN accelerators [21] offer a great opportunity to break the memory wall by utilizing their special architectural advantages. nvCiM architectures reduce data movement with an in-situ weight data access scheme [42]. Emerging NVM devices (e.g., RRAMs, STT-RAMS, and FeFETs) are utilized so that nvCiM platforms can achieve higher energy efficiency and memory density compared with traditional MOSFET [39] based designs. More specifically, nvCiM can achieve low latency and high energy efficiency because, (1) the CiM structure avoids the long latency for moving data across multi-level memory hierarchies to retrieve the intermediate data and/or DNN weights; (2) analog computing engine performs dot-product in a compact manner, thus reducing the amount of intermediate data (i.e., partial sums) generated in multiply-and-accumulate (MAC) operations; (3) the crossbar structured matrix-vector multiplication (VMM) engine offers high parallelism that can perform VMM in one CiM cycle, thus shortening the latency of DNN operations.

However, such accelerators suffer greatly from design limitations. Firstly, because of emerging NVM devices tend to have low precision (1–4 bits) of (i.e., single NVM device can only represent 1 to 4-bit data and more than one device is needed to represent data in higher precision) and the limit of chip area, weight precision of neural networks mapped to nvCiM accelerators is limited. Secondly, most nvCiM accelerators require digital-to-analog converters (DACs) to convert the digital input data to analog signals so that it can be processed in crossbars and also analog-to-digital converters (ADCs) to convert the computation results back to digital signal for other neural network operations (e.g., activation and normalization). The precision of intermediate activation data is limited by the precision of DACs and ADCs. Thirdly, NVM devices, ADCs, and DACs suffer from device-to-device variations

due to manufacture and programming defects and cycle-to-cycle variations due to the computational environment difference. Compared with their digital counterparts that can tolerate such noises, because of the analog nature of nvCiM accelerators, the calculations performed in these platforms are not noise-free. The noisy nature of the computations leads to performance degradation that (1) models deployed on nvCiMs typically gets lower accuracy than their ideal counterparts trained in the data centers and (2) developers will not be able to know the exact accuracy of a model before it is deployed on a certain copy of the nvCiM product.

This reliability issue and its impact on DNN performance have been studied from different levels of design, including behavioral level explorations [15, 50], architecture level analysis [49], and device-level observations [56]. Cross-layer co-design efforts that simultaneously explore DNN model and hardware design pairs that can together achieve both high perception task performance and desirable hardware reliability are the current direction of this field [23].

In this chapter, we focus on design efforts targeting crossbar-based nvCiM DNN accelerators. We first introduce three typical emerging NVM devices including resistive random access memory (RRAM), ferroelectric field-effect transistor (FeFET), and Spintronics (STT) Devices. We then describe typical nvCiM DNN accelerator designs, their key components, and their benefits. After that, we discuss the limitations of nvCiM DNN accelerators and some key findings for these limitations. Finally, we introduce methods proposed to address the unreliability issue of nvCiM DNN accelerators from three aspects, encoding, DNN model training, and DNN architecture selection.

2 Non-volatile Devices

2.1 RRAM

Resistive random access memory (RRAM) is a two-terminal device that can be programmed into different levels of resistance value by using programming voltages in different magnitudes and duration.

As shown in Fig. 1a, the major component of RRAMs is a metal-insulator-metal (MIM) stack, where a dielectric layer is stacked in the middle of two electrode layers. When provided a programming voltage, a filamentary path, also called conductive filament (CF) [19], is created by soft electrical breakdown or forming in the electrode layers. In this filamentary path, a large concentration of defects, e.g., oxygen vacancies in metal oxides [4] or metallic ions injected from the electrodes [32], are then driven by field-induced migration and diffusion. Application of a positive voltage to the top electrode, where the defects are concentrated, induces defect migration towards the bottom electrode, thus causing the transition to the low-resistance state (LRS), because conduction is enhanced at defect sites. Application of a negative voltage, to the contrary, induces defect migration back to the top electrode, thus

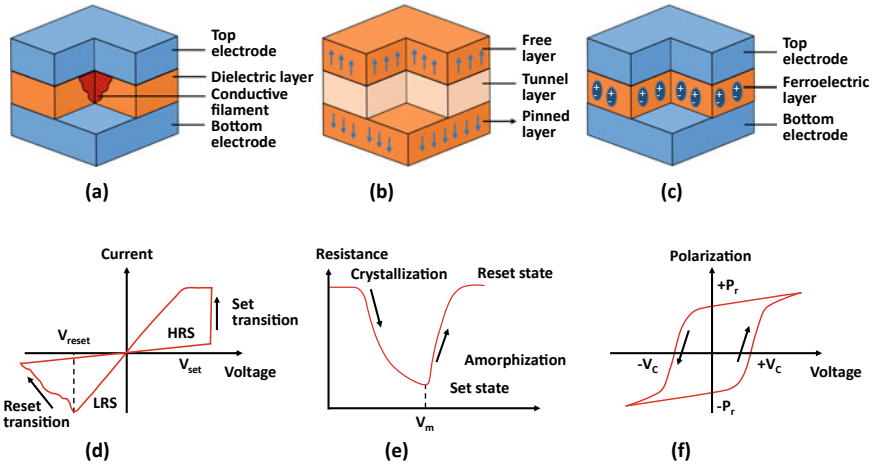

Fig. 1 Illustrations for key structures of different emerging NVM devices and their characteristics [8]. **a** RRAMs, **b** MRAMs and **c** FeFETs. The actual devices are more complex then these illustrations. **d**–**f** Are their characteristics, respectively

causing the transition to the high-resistance state (HRS) due to the disconnection of the CF. These transitions can be seen in the idealized current–voltage (I–V) characteristic in Fig. 1d, where the transition to the LRS (set operation) and the transition to the HRS (reset operation) occur at opposite voltages. Similar to the bipolar RRAM concept shown in Fig. 1d, unipolar RRAMs have also been presented, where the set and reset processes both occur under the same voltage polarity because of the dominant role of Joule heating in creating and dissolving the CF [24, 51]. All of these devices rely on the diffusion and migration of defects and will be referred to as RRAM throughout this chapter.

RRAM is a promising technology for in-memory computing thanks to the key features discussed below. First, its resistance ratio between HRS and LRS (on/off ratio) is generally greater than ten, which allows a clear distinction between digital '0' and '1'. This feature can be further exploited by dividing this gap between HRS and LRS in a non-binary manner, i.e., into multiple levels, resulting in a multi-level device that can represent multiple bits of data. This helps RRAM to offer a high-density storage scheme. Secondly, RRAM can operate at a moderately high switching speed (typically below 100 ns and some devices can achieve even in the sub-ns regime [11, 34]). Thus, RRAMs can operate in platforms with high clock speeds. Finally, RRAM is more durable compared to conventional flash storage devices [28]. This makes training DNNs on RRAM-based platforms possible.

2.2 Spintronics Devices

The spintronics devices are two or three-terminal devices equipped with a magnetic tunnel junction (MTJ) that stores information using magnetization direction of its recording layer and utilizes tunnel magnetoresistance (TMR) effect for reading where the resistance of the MTJ changes according to the stored information. In this section, we introduce the two-terminal version of this device, named spin-transfer torque (STT) device. When used as a programmable memory, this kind of device is also called Spin-transfer torque magnetic RAM (STT-MRAM).

Figure 1b shows a magnetic tunnel junction (MTJ), which is the major building block for most Spintronics Devices. The MTJ consists of a MIM structure where two ferromagnetic metal layers are divided by a thin tunnel oxide. An example of ferromagnetic metal materials used in MTJs is the CoFeB alloy, and an example material for the tunnel oxide is MgO. For the two ferromagnetic layers, one is referred to as the pinned layer and the other as the free layer. The magnetic polarization of the pinned layer is structurally fixed so that it can act as a reference point. On the other hand, the magnetic polarization of the free layer can be modified by a programming procedure.

Depending on the state of the free layer, the two ferromagnetic polarization can thus be either to the same direction (parallel) or to the opposite direction (antiparallel). Parallel polarization of the two layers puts the device into a low resistance state (LRS), and antiparallel means a high resistance state (HRS) due to the tunnel magnetoresistive effect [7]. Researchers are working on finding efficient ways to flip the state of the MTJ and the spin-transfer torque (STT) is one of the newer and more competitive candidates to offer a scalable and low-efficient flip [33]. In the STT procedure, transition to the parallel state takes place directly by conduction electrons, which are first spin-polarized by the pinned layer, then rotate the magnetic polarization of the free layer by magnetic momentum conservation [41]. Similarly, the free layer magnetization can be rotated to the antiparallel state by applying an opposite voltage (hence opposite current direction). The relative difference in resistance of the LRS and HRS, also called the magnetoresistance ratio when referring to spintronics devices, is typically around 200% [53]. STT-based devices are also fast, with a switching speed typically lower than 1 ns, and durable, with an endurance above 10^{14} [6].

In STT devices, STT induced magnetization switching [5, 41] is used to store data in to the device (write process). Its primitive cell has one cell transistor and one MTJ (1T1MTJ), which can achieve a relatively small cell size of ideally $6F^2$, where F is the feature size of the MTJ layer. The write current passes through the tunnel barrier, as is also the case with the read current. Accordingly, the read current should be small enough so that the write event, i.e., magnetization switching, does not take place, and the write current should be small enough that it does not give rise to a barrier breakdown.

2.3 FeFET

A ferroelectric transistor (FeFET) is a three-terminal device equipped with a layer of ferroelectric (FE) material. It can either be configured to a steep switching mode to serve as an efficient FET or a non-volatile (NV) mode to serve as a programmable switch.

The structure of a FeFET is similar to a regular bulk MOSFET or FinFET, except that in its gate stack, there is an additional layer of ferroelectric (FE) material. Besides this FE material, a metal layer between the FE and dielectric may or may not be included [2]. Designs of FE transistor structures with [29] and without [40] this layer both demonstrate state-of-the-art efficiencies. It is worth noting that although some FE materials (e.g., hafnium zirconium oxide (HZO)) are both efficient and highly compatible with CMOS processes and can thus be realized on the industrial scale, other FE materials (e.g., lead zirconium titanate (PZT) [3]) may be incompatible with CMOS processes.

As discussed above, FeFETs can operate in two different modes: an NV mode or a steep switching mode. Basic structures of FeFETs in these two modes are the same, except that in different configurations (e.g., material thickness, gate length, and width), the relative capacitance of the FE material and the underlying FET changes, resulting in different modes of operation. In this chapter, we discuss the properties of FeFETs in the NV mode because FeFETs used in nvCiM DNN accelerators are majorly in this mode.

NV mode of FeFETs are discovered later than its steep switching counterpart at the emergence of HZO-based FeFETs [31]. The non-volatile property results from the hysteretic polarization (P) versus voltage of the FE material (V_{FE}) shown in Fig. 1f. When the FE material is placed in series with the gate of a transistor, the hysteretic window of P versus V_{GS} is reduced because the MOS structure of the FET and the associated depolarization fields imposes a capacitance and the total capacitance between gate and source changes [44]. Nevertheless, a sufficiently thick FE broadens the hysteretic window so that the hysteretic behavior is preserved and can be observed in the $I_D - V_{GS}$ transfer characteristics of this device Fig. 1f. This corresponds to the non-volatile, hysteretic mode of FeFETs. In this mode of operation, at $V_{GS} = 0V$ (i.e., when the supply voltage is turned off), the FeFET exhibits two stable states which correspond to positive or negative polarization retention in the FE layer. For an n-type FeFET, the device exhibits high resistance states (HRS) when $P < 0$ and low resistance states (LRS) when $P > 0$. For a p-type FeFET, it is in HRS when $P > 0$ and LRS when $P < 0$. Thus, when the FE layer is sufficiently thick, non-volatility can be embedded inside a transistor, i.e., FeFET can operate as an NV memory and a transistor switch at the same time.

3 CiM DNN Accelerators

3.1 Computing-in-Memory

Conventional von-Neumann architecture is not efficient because the cost of data movements between memory and processing units is high. This issue is called *the memory wall*. More seriously, the technologies for logic units are growing faster than memory cells, causing a significant gap between computation and memory access. Thus, various efforts have been made to break *the memory wall* by moving the computations closer to memory. The integration of memory and computation is an evolving concept and is developing along with technological advances [16]. We first introduce an earlier concept which is now considered near memory computing (NMC). Researchers embed processing cores into dynamic random-access memory (DRAM) modules [12, 35, 37] so that data can be processed in the DRAM module. This avoids sending data from DRAM to CPUs across the complex memory hierarchy. However, integrating DRAMs and processing units on the same chip is not beneficial if the communication cost between memory and processing units is not reduced. The concept of 3D stacking is adopted to address this issue. By stacking multiple silicons on top of each other and utilizing through-silicon-vias (TSVs) to handle inter-silicon layer communications, 3D stacking allows the processing unit to be integrated as additional layers of the stacked chip and can provide higher bandwidth compared with putting memory and logic in different chips [13, 18, 55]. However, these methods do not actually use memory modules for data processing and are still sending data from memory to logic.

A step further from NMC is computing-in-memory (CiM), where processing is directly performed inside the memory array. The latency and energy efficiency requirements of edge devices greatly inspired researches in this field. The integration of processing and memory units can be done in different levels of granularity. The extremest design of CiM is that each of the memory cells is able to perform logic operations [26]. This is referred to as fine-grained CiM. There is also a spectrum of designs between fine-grained CiM and NMC. A typical design is to empower memory arrays (of SRAM or DRAM) with processing abilities so that data can be processed inside operations inside and between memory arrays. This can be achieved by modifying the peripheral circuitry of these memory arrays. This approach is referred to as coarse-grained CiM.

The CiM concept is further evolved with the help of new advances in emerging NVM device technologies. Specifically, NVM devices including RRAMs, STT-MRAMs, and FeFET-based RAMs can offer high density, good scalability, and high power efficiency. Thus, these devices are natural replacements for SRAMs or DRAMs in CiM architectures. Various recent efforts utilize CiM-capable NVM devices instead of SRAMs or DRAMs as building blocks of either cache or main memory. One direction of research is to use NVM simultaneously as storage and logic devices by re-designing sense amplifiers so that NVM arrays can perform a subset of logic and arithmetic operations [22, 30, 38]. Another direction is to use

NVMs to build content-addressable memories (CAMs). CAMs can perform searches in a parallel manner, thus reducing the search time significantly. Moreover, search in CAMs requires little data movement, which leads to low energy consumption. The third direction is to use NVM devices to build DNN accelerators. These accelerators can directly execute matrix-vector multiplication inside the memory array. This saves the cost of data movements. The advances of NVM-based CiM DNN accelerators are discussed in detail in the following sections.

3.2 *Crossbar-Based Vector-Matrix Multiplication Engine*

Crossbar array is the key component of nvCiM DNN accelerators. As shown in Fig. 2, a crossbar array can be considered as a processing element for matrix-vector multiplication where matrix value (i.e., weights for DNNs) are stored at the cross point of each vertical and horizontal line with resistive NVM devices such as RRAMs and FeFETs, and each vector value is propagated through horizontal data lines. In this work, we mainly introduce an RRAM-based design. Designs using other kinds of NVM devices are with similar structures. The calculation in crossbar array is performed in the analog domain but additional peripheral digital circuits are needed for other key DNN operations (e.g., non-linear activation and pooling), so DAC and ADCs are adopted between different components.

As is demonstrated in Fig. 2, every bitline (vertical) is connected to every word-line (horizontal) via NVM cells [39]. Assume that the cells in the first column are

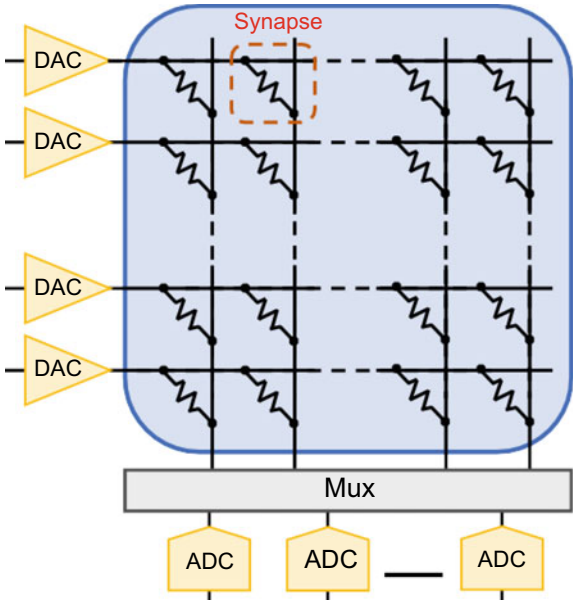

Fig. 2 Illustration of crossbar array architecture. The input is fed horizontally and multiplied by weights stored in the NVM devices at each cross point. The multiplication results are summed up vertically and the sum serves as an output

programmed to resistances $r_1, r_2, \ldots, r_n$, where n is the number of rows. The conductances of these cells, $g_1, g_2, \ldots, g_n$, are the inverses of their resistances ($g_i = 1/r_i$). If voltages $V_1, V_2, \ldots, V_n$ are applied to each row 2, cell i generates current V_i/R_i, which is equivalent to $V_i \times g_i$, into the bitline, based on Kirchoff's Law. The total current accumulated on the bitline is the sum of currents passed by each cell in the column, i.e., $I = \sum_{i=1}^{n} V_i \times g_i$. This current I represents the value of a dot product operation, where one vector is the set of input voltages at each row $\mathbf{V}$ and the second vector is the set of cell conductances $\mathbf{g}$ in a column, i.e., $I = \mathbf{V} \cdot \mathbf{g}$.

As shown in Fig. 2 $\mathbf{V}$ is applied to all columns in parallel. The currents emerging from each bitline can therefore represent multiple vector-vector dot product, which is then a vector-matrix multiplication. VMM is the key operation of DNNs. In a fully connected layer, for example, there are multiple neurons and each neuron is fed with the same input vector, but each of the neurons has a different set of synaptic weights. This operation can be represented by $\mathbf{O} = \mathbf{V}\,\mathbf{G}$ where $\mathbf{V}$ is the input, $\mathbf{G}$ is the weight matrix for neurons and $\mathbf{O}$ is its output. The crossbar array shown in Fig. 2 represents an $n \times m$ crossbar array that performs dot products on n-entry input vectors for m different outputs in a single CiM cycle.

Note that the result of the VMM operation would also need to be applied a bias value and passed through a non-linear activation function. This is done off the crossbar array. Thus, peripheral circuits are needed to perform these operations. Moreover, crossbar arrays handle VMM operations in the analog domain while other peripheral circuits are digital. DACs and ADCs are needed to transform data to and from the analogy domain. Generally, for each row of the crossbar array, there is a dedicated DAC to serve this wordline. However, ADCs are large in area and power-hungry. Thus, multiple bitlines need to share one ADC and this is achieved by the sense-and-hold circuits along with the MUX selector.

3.3 General Architecture of nvCiM DNN Accelerators

Various accelerator architectures have been proposed to utilize the nvCiM crossbar arrays for more efficient DNN acceleration. There are generally two fashions of acceleration, one only accelerates the inference path of DNN models and the other also considers DNN training acceleration. In this chapter, we focus on DNN inference acceleration and we introduce two well-known architecture level designs, ISAAC [39] and PRIME [10] for this scheme.

The first design, In-Situ Analog Arithmetic in Crossbars (ISAAC) [39] uses crossbar arrays for both DNN weight storage and processing elements for VMM operations [54]. As shown in Fig. 3, ISAAC is implemented with a hierarchical-structured architecture whose major component is "tile". Each tile consists of multiple in-situ MAC units (IMA), eDRAM buffers, and key DNN circuitries including shift-and-add (SA), sigmoid, and max-pooling units. Thus, a tile can perform DNN operations individually. Each IMA unit is equipped with a few crossbar arrays and ADCs connected by a shared bus. Different from traditional SRAM-based designs, writing NVM devices

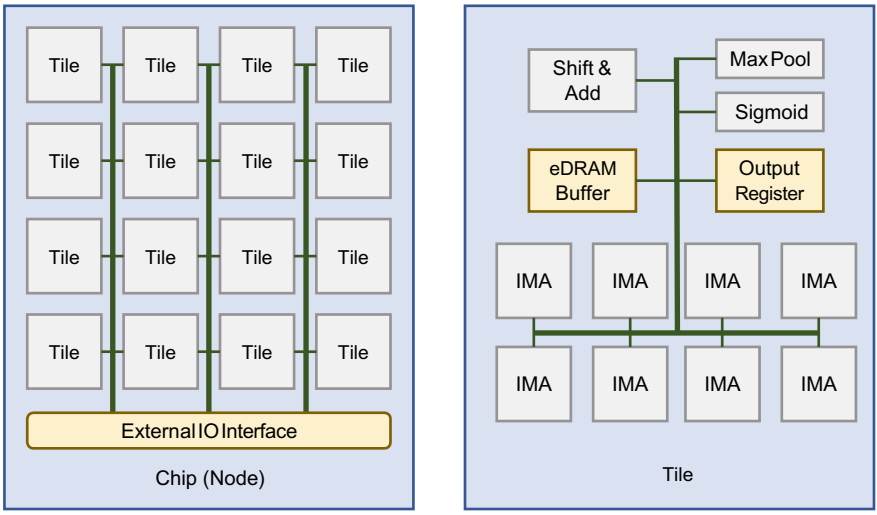

Fig. 3 Illustration of ISAAC architecture. ISSAC is composed of a group of tiles and each tile consists of multiple crossbar-based IMAs, buffers, and peripheral circuits for other key DNN operations

is expensive (both in terms of time and energy consumption), so re-configuring crossbars in runtime are not feasible and thus crossbar arrays cannot be reused and each array is dedicated to only one CNN layer. The outputs of a former layer are temporarily preserved in the eDRAM buffer so that they can be used as the input of the next layer. Note that, except for the structure inside a "tile", the architecture of ISAAC is very similar to its digital DNN accelerator counterpart DaDianNao [9], which is a state-of-the-art architecture when ISAAC is proposed. After tape out, the researchers show that, with a 16-chip configuration, ISAAC achieves 14.8× higher throughput while consuming 5.5× lower energy than DaDianNao. This means (1) ISAAC can achieve higher energy efficiency than state-of-the-art and (2) crossbar array-based design is a key contributor to this efficient design.

Different from ISAAC that never re-configures NVMs, the PRIME architecture [10] uses a scheme where a portion of the NVM arrays can alternate between storage and compute units during runtime. As shown in Fig. 4, the authors modify the standard wordline decoder and drivers (WDD), column multiplexers, and sense amplifiers so that they can better suit the RRAM-based crossbar arrays, and configure the storage banks into three different function units, memory subarrays (MS), full-function subarrays (FFS), and buffer subarrays (BS). The FFS is the key component that can alternate from memory to computational units. In the computation mode, FFS can perform VMM for DNNs, and in the storage mode, FFS buffers the intermediate data generated by VMMs. Similarly, the BS also acts as storage when FFS is not in computation mode. The sense amplifier is reconfigured to detect the higher precision analog value for computation compared to storage requirements so that matrix multiplication can be performed. The modified column multiplexer executed

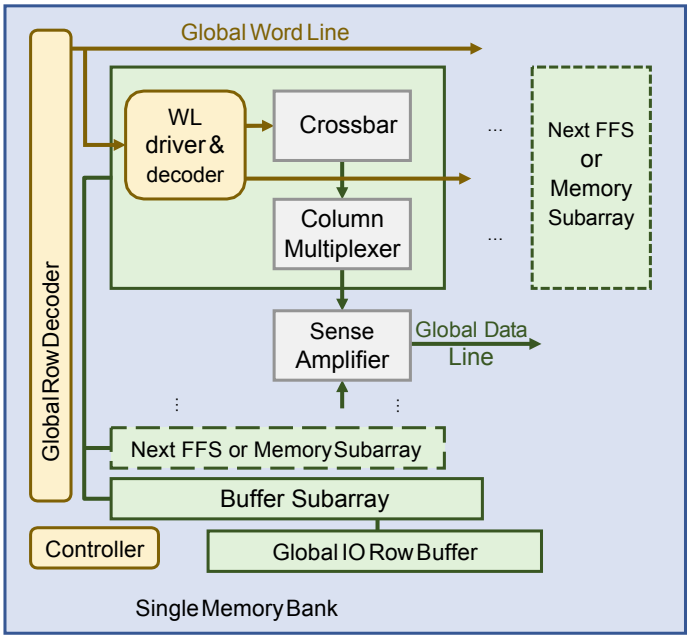

Fig. 4 Illustration of the prime memory bank. Each FFS can operate in two modes, one is computation mode for MAC operation of DNNs, and the other is storage mode buffering and data preservation

analog substractions and nonlinear threshold functions. Although their implementation exerted a 60% area overhead, the computation energy was saved by 94% by reducing external memory accesses.

4 Device and Circuit Non-idealities

Although nvCiM can offer low latency and high energy efficiency, there are two major limitations of nvCiM, low data precision, and low device reliability. For the first issue, due to the limitation of the area and power budget, both the weight stored in the NVM devices and intermediate activation data can not be represented in a high precision manner. nvCiM DNN accelerators generally use data representations of four to eight bits [10, 39]. This problem is similar to the quantization problem of the traditional digital DNN accelerators and has been sufficiently discussed [17, 45]. However, the origin, simulation method, and mitigation approach of the reliability issue of nvCiM DNN accelerators are still open questions and are still receiving heated discussions. In this section, we introduce the origin of the reliability issue of nvCiM DNN accelerators with an example of RRAM devices. For STT and FeFET

devices, the source of unreliability is similar but the significance and specific behavior of these noise sources are slightly different.

Various research about developing fault models for RRAM and other emerging NVM devices has been established. In this section, we focus on five noise sources that are directly related to the unreliability of nvCiM DNN accelerators: thermal noise, shot noise, random telegraph noise (RTN), programming errors, and endurance failures [14].

4.1 Thermal Noise

Thermal noise is also known as Johnson-Nyquist noise. It is electronic noise caused by the thermal agitation of carriers and is a property of all passive devices. It happens regardless of whether a voltage is applied to the device. A well-established model for thermal noise is by placing a current source in parallel with the ideal target device. The current source is also known as the noise current and its magnitude is modeled by a Gaussian distribution with zero mean and a standard deviation of $\sqrt{\frac{4K_B T \Delta f}{R}}$, where K_B is the Boltzmann constant ($\approx 1.38 \times 10^{-23}$ J/K), T is the temperature in Kelvins, δf is the bandwidth of the signal measured, and R is the resistance of the ideal target device. Thermal noise is a fundamental property of resistive circuit elements. From the model, we can observe that the only way to reduce thermal noise is to reduce the device temperature. To handle this source of noise, noise resilient architectures that can operate under thermal noise need to be devised.

4.2 Shot Noise

Shot noise is also a fundamental source of noise caused by the physical nature of electronic devices. This source of noise is called Poisson noise because it can be modeled by a Poisson process. The key cause of shot noise is the discrete nature of currents where electric currents actually consist of flows of discrete charges (e.g., electrons). When the number of electrons flowing through the device at a certain point of time fluctuates, a fluctuation of current through a device can be observed. This can affect the measurement accuracy when a detector is sensing the current flowing into it. Although shot noise is easy to be averaged out provided enough measurement time, devices working in high frequencies (e.g., nvCiM DNN accelerators) still suffer from such noise. As discussed above, a Poisson process is a more precise way of modeling shot noise, but this noise model is too complex when embedded in other models. A simpler model is a zero-mean Gaussian noise with a standard deviation of $\sqrt{2qI\Delta f}$, where q is the charge of an electron ($\approx 1.6 \times 10^{-19}$ C), I is the current flowing through the ideal target device, and Δf is the bandwidth of the signal measured.

4.3 Random Telegraph Noise

Random telegraph noise (RTN) exists in both CMOS and emerging NVM device circuits but is considered as a major cause of faults of emerging NVM devices [20]. RTN is also called burst noise and is caused by the charge carriers that are temporarily trapped inside the device, thus changing the effective resistance of the device. The result is a temporary and unexpected reduction in the resistance of a device at runtime. The trapping and untrapping of the charge carrier is modeled mathematically by means of the telegraph process, which is a Markovian continuous-time stochastic process that jumps discontinuously between two distinct values.

4.4 Programming Errors

Programming errors refer to the difference between the actual device resistance and the target resistance due to the non-ideal configuration of the device. This is generally caused by both the process variations and temporal variations of each device. Affected by the former noise, when applied the programming voltage of the same magnitude and duration, the resistance of different instances of emerging NVM devices can be different. The latter leads to the fact even when applied to identical programming pulses, an NVM device can be programmed to different values in different trials of programming. A complex but effective way to mitigate this issue is to use a scheme called write-and-verify [1, 36, 47]. The key operation is to iteratively apply a series of short pulses (write) and then check the difference between current and target resistance (verify), converging progressively on the target resistance. In deploying accelerators for Neural Network inference, this time-consuming progress is tolerable because once programmed, no more modifications to the resistance are needed during the entire life span of the accelerator. This scheme pulls down the programming error to less than 1%. This 1% of error can be modeled by a zero-mean Gaussian noise where the standard deviation is determined by the error upper bound of the write-and-verify process.

4.5 Endurance and Retention

Endurance Failure is about the device being able to preserve their property after multiple times of write operations or and retention is about being able to read the desired data at a long period of time after programming. The endurance of emerging NVM devices varies widely based on the material properties and write mechanisms. The typical endurance for CMOS-based SRAM is 10^{16} which means typically, after this amount of write, the device would be stuck at a certain value, and writing it would be infeasible. The typical endurance for STT-MRAM is 10^{15}, for FeFET is

10^5 and for RRAM is 10^7 [14]. On the other hand, being able to read out the correct information when it is a long period of time after the device is programmed is also an important subject. This is called the retention issue. For simple CiM implementations like Memristive Boltzmann Machine, a typical worst-case lifetime is 1.5 years, but for nvCiM DNN accelerators, the system is more complex and the lifetime is shorter. To mitigate the effect of the endurance issue, researchers proposed a fault-tolerant online training method [46] that maps the weight matrices stored in crossbars for computation around faults or endurance failures through a combination of neural network pruning and data remapping. This scheme increases the life of the neural network accelerator, allowing it to be used for training.

5 Impact of Device Variation on DNN Acceleration

5.1 Model of Device Variation

The source of device variations and their behaviors are introduced in Sect. 4, but modeling such device characteristics is not a simple task. A straightforward way is to abstract the behavior of different devices into circuit-level models [56] and utilize circuit-level simulation tools (e.g., *SPICE*) to investigate the behavior of certain nvCiM DNN accelerators. However, because of the complexity of both neural network typologies and DNN accelerator architectures, building circuit-level models for nvCiM accelerators requires great human effort and needs to be modified each time a new type of accelerator architecture is proposed. Moreover, circuit-level models are computationally intensive. Using such models to simulate complex DNN accelerators requires considerable evaluation time and is not suitable during design phase explorations. Thus, a simple and effective model for the impact of device variations is needed.

One of the effective modeling methods is to model the device variation as a whole and use a Gaussian distribution to represent it [14, 15, 23, 50]. Here we introduce one representative modeling method [15] using Gaussian variables.

The NVM device electrical property, e.g., conductance, is subject to the combined effect of different variation sources as in Sect. 4. The actual conductance values g considering variations on n devices of a crossbar array can be written as:

$$\mathbf{g} = g_{0,n\times 1} + \Delta g_g + F\left(g_{0,n\times 1,\mathbf{r}} \approx \overline{g_{0,n\times 1}} + f\left(g_{0,n\times 1}, \mathbf{r}\right)\right) \tag{1}$$

where $\overline{g_{0,n\times 1}} = g_{0,n\times 1} + \Delta g_g$ with $g_{0,n\times 1}$ denoting the expected conductance and Δg_g denoting the global conductance variation as a constant for all the devices on the same die; $\mathbf{r}$ models the underlying spatially correlated and dynamic variations; $f(g_{0,n\times 1}, \mathbf{r})$ is a function describing the dependence of variations on the expected conductance and can be approximated by $f(g_{0,n\times 1}, \mathbf{r})$ due to the relatively small value of variations w.r.t. the nominal values [11].

Since the mapped weights $\mathbf{w}$ are linearly related to conductance as $\mathbf{w} = c_1 \times \mathbf{g} + c_0$, where c_1 and c_0 are two constants, each weight w_i represented by multiple devices can be modelled as a Gaussian variable:

$$w_i = \mathcal{N}\left(u_{0,i}, \Psi\left(u_{0,i}\right)^2\right) \tag{2}$$

$$= \mathcal{N}\left(c_1 \overline{g_{0,i}}, +c_0, c_1^2 f\left(\overline{g_{0,i}}\right)^2 \left(\sum_{k=1}^{m} \lambda_{i,k}^2 + \lambda_{i,n}^2\right)\right) \tag{3}$$

5.2 Impact of Device Variation on DNN Outputs

After finishing modeling the device variations, we can then investigate the impact of device variations on nvCiM DNN accelerators. A typical study is to evaluate such impact on an accelerator targeting image classification tasks [49]. In this section, we introduce the findings of the authors of [49].

A starting point is understanding the effect of device variations on the output of a DNN model. The forward path of a DNN model can be viewed as a function of the input and the weight value of the model. Formally speaking, a DNN inference process can be defined as:

$$\mathbf{O} = F(W, \mathbf{I}) \tag{4}$$

where F is the DNN architecture, W is the DNN weights, $\mathbf{I}$ is the input vector, and $\mathbf{O}$ is the output vector.

In classification tasks, the output vector $\mathbf{O}$ for each input (not batched) is a 1-D vector whose size is the number of possible classes. Each element of this vector represents the model's confidence that the input images should be classified into a certain class. Thus, the class with maximum value in $\mathbf{O}$ is what the model predicts to be the best choice for classification. During training, $\mathbf{O}$ is passed through a *Softmax* function so that the confidence for each class is between 0 and 1 and the sum of confidences among different classes is 1. However, *Softmax* is not necessary during DNN inference because it does not change the order of the values in $\mathbf{O}$. The final predicted class of $\mathbf{I}$ is calculated by *argmax* ($\mathbf{O}$), which is the index of the item in $\mathbf{O}$ that has the maximum value. As we focus on inference, the vanilla version of $\mathbf{O}$ before *Softmax* is the key.

Taking device variation into account, a model deployed on nvCiM DNN accelerators can be represented as:

$$\mathbf{O}_{Dep} = F(W_{Dep}, I) = F(\mathcal{N}(W_{Exp}, \sigma), I) \tag{5}$$

where W_{Dep} is the weight actually deployed on the accelerator and according to Eq. 2, it can be modeled as a Gaussian variable whose mean is W_{Exp}, which is the trained value of the neural network to be deployed, and the standard deviation is σ, which can be calculated using Eq. 2. $\mathbf{O}_{Dep}$ is the affected output.

One indicator of the effect of device variations on nvCiM accelerators is the difference in output. Formally speaking, we can define *output change* as the difference between the output without device variation and the output value under the impact of device variation:

$$\mathbf{O}_{Change} = F(W_{Exp}, I) - F(\mathcal{N}(W_{Exp}, \sigma), I) \tag{6}$$

Note that $\mathbf{O}_{Change}$ is also a random variable.

In order to get a glance at the statistical behavior of $\mathbf{O}_{Change}$, according to the workflow introduced in Sect. 5.2, the authors train a LeNet model for the MNIST dataset [27] to state-of-the-art accuracy. The authors then randomly choose one input image in the test dataset and sampled 10k different instances of noise. With this setup, the authors gathered 10k different $\mathbf{O}_{Change}$ vectors.

For MNIST, $\mathbf{O}_{Change}$ is a vector of 10, with each element representing the confidence of classifying the input image into one certain number digit. Because a high-dimensional vector is not a good choice for analytical study and visualization, each element of these vectors is visualized independently, so 10 instances of distribution data are collected.

Each element of $\mathbf{O}_{Change}$ follows Gaussian distribution. To visualize this finding, the authors plot the histogram of the distribution of each element of $\mathbf{O}_{Change}$ vector and the corresponding Gaussian distribution that fits it. The visualization result for the first element of $\mathbf{O}_{Change}$ is shown in Fig. 5. It is obvious that the visualized variable is Gaussian.

This observation generalizes in various networks in various datasets. For the MNIST dataset, three models are analyzed: (1) LeNet and two-layer-multilayer perceptrons (2-layer-MLP) using (2) ReLU and (3) Sigmoid activation. For the

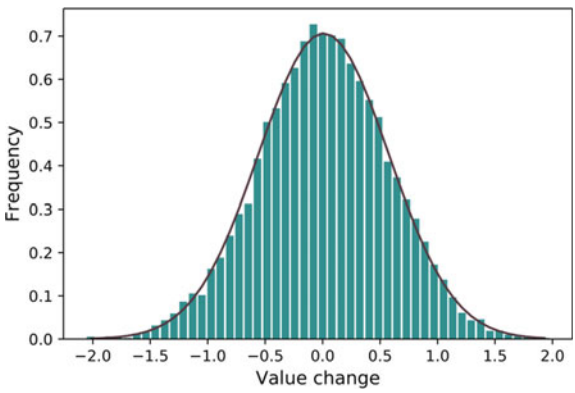

Fig. 5 $\mathbf{O}_{Change}$ distribution of LeNet for MNIST. 10k $\mathbf{O}_{Change}$ vectors are gathered from one trained LeNet model affected by 10k different instances of weight values from $\sigma = 0.04$. This figure shows the distribution of the first item of the gathered $\mathbf{O}_{Change}$ vectors. It is obvious that the visualized variable is Gaussian

CIFAR-10 dataset [25], the authors of [49] test four models: (1) a conventional floating-point CNN, (2) a quantized CNN, and two ResNets, (3) ResNet-56 and (4) ResNet-110. For each model, three different initializations are used to train three different sets of weights.

The authors of [49] collect all $\mathbf{O}_{Change}$ variables and find the closest Gaussian variable that fits each of them. To measure the similarity of $\mathbf{O}_{Change}$ and its Gaussian counterpart, two widely used standards: mean square error (MSE) and Chi-square (χ^2) test are used. For variables with one element, MSE can be described as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(O_i - E_i)^2 \tag{7}$$

and χ^2 test can be depicted as:

$$\chi^2 = \sum_{i=1}^{N}\frac{(O_i - E_i)^2}{E_i} \tag{8}$$

where O_i and E_i are the observed ($\mathbf{O}_{Change}$) and estimated (Gaussian) value of, normalized in the form of probability density, and N is a user-defined granularity. Here $N = 100$ is used because it is precise enough when there is a total of 10k instances of $\mathbf{O}_{Change}$ data. The similarity of a vector is averaged out among all of its elements and the final similarity is also averaged out among all different initializations.

The similarity of $\mathbf{O}_{Change}$ distribution and its Gaussian fit for different models are shown in Table 1. For each model tested, the average χ^2 test results among different initializations are all below 0.1 and MSE are all below 10^{-3}, which indicates we can have high confidence that $\mathbf{O}_{Change}$ distribution is Gaussian. Moreover, this observation is scalable because, for both extremely shallow (e.g., 2-layer MLP) and very deep (RestNet-110) candidates, both errors do not increase. Thus this observation generalizes across different DNN models targeting classification tasks. With this

Table 1 The similarity of $\mathbf{O}_{Change}$ distribution and its Gaussian fit for different models

Model	Dataset	χ^2 (10^{-2})	MSE (10^{-4})
MLP-ReLU	MNIST	5.22	3.20
MLP-Sigmoid	MNIST	5.81	2.20
LeNet	MNIST	4.59	2.67
Float-Conv	CIFAR-10	7.01	3.03
Fixed-Conv	CIFAR-10	6.79	2.74
ResNet-56	CIFAR-10	4.56	1.79
ResNet-110	CIFAR-10	4.81	2.01

The χ^2 test result and MSE between the $\mathbf{O}_{Change}$ and its Gaussian fit counterpart is presented. Both tests show that the $\mathbf{O}_{Change}$ is a multi-dimensional Gaussian variable w.r.t. different instances of noise

conclusion, the authors of [49] claim that, **with any independent and identically distributed Gaussian noise on weight, the output vector of the same input image follows a multi-dimensional Gaussian distribution**[1] **over different samples of noise**.

This claim is very strong and there is only empirical support for it. However, it is not counter-intuitive. The output of the first convolution layer is the summation of the multiplication result of deterministic inputs and Gaussianly distributed weights and is thus a summation of Gaussian distributions. The summation of Gaussian variables is also a Gaussian variable, so the output of the first layer is a Gaussian variable. After activation, the input of the second layer is a transformed Gaussian variable. After propagating through this layer, each output value is the sum of multiple multiplication results, and operands for each multiplication are both Gaussian variables. It is also worth noticing that, for the same layer, the standard deviation σ for each noisy weight is the same. So the results of each multiplication are close to IID and with enough number of operands for this summation, the accumulated variable can be approximated by Gaussian variables. Thus, although the final output may not strictly be a Gaussian variable, a Gaussian approximation can be observed.

6 Dealing with Device Non-idealities

The majority of noise sources of nvCiM DNN accelerators are random noise that is difficult to eliminate during device production. Fortunately, there are opportunities from the accelerator architecture, DNN topology design, and DNN training aspects that can help to mitigate the effect of device variations. In this section, the authors introduce four different efforts from these three aspects.

6.1 Error Correction

As discussed in Sect. 3.3, nvCiM accelerators process DNN models in a layer by layer manner and devise nvCiM processing units that consist of crossbar arrays and other peripheral digital blocks to perform matrix-vector multiplication and other key DNN operations including non-linear activation and pooling. From the accelerator architecture design aspect, it is a straightforward idea to equip nvCiM platforms with error correction abilities so that they can mitigate the effect of device variations.

In this section, we introduce one representative work [14] that uses error correction code to assist nvCiM computation. The authors use a group of arithmetic codes, named AN-codes [43] for error correction. Arithmetic codes are a class of error correction codes (ECCs) that can preserve the result of arithmetic operations with noisy operands. AN-codes are a set of arithmetic codes that apply arithmetic weight

[1] Note that each element of the output are deeply co-related, not independent.

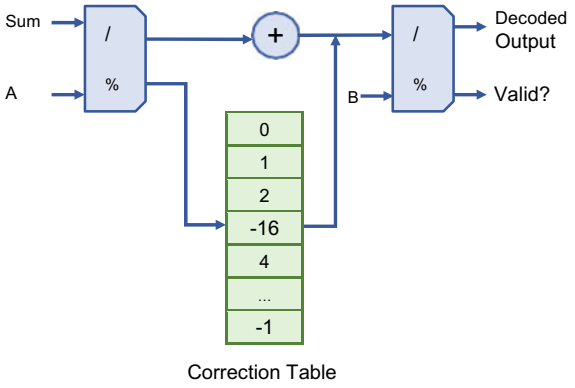

Fig. 6 Illustration of error correction unit circuitry. This is a lookup table styled design

to each operand so that it can maximize the arithmetic distance between codewords. An example of AN codes that utilizes residues is, for a given integer K and operands A and B, $KA + KB = K(A + B)$ and $(KA + KB)\%\ K \equiv 0$. The ECC units can detect and correct the error according to the residue.

The error correction unit (ECU) in [43] has three major components: two divide/residual units for the residual computation of A and B (one each), and a correction table that maps each residual to a syndrome. The output of the first divide/residual unit computes the integer division of the input by A and outputs the residual along with the quotient. The residual is used to index into the correction table, and the value read from the correction table is added to the result. This value is then fed into the second divide/residual unit where it is divided by B. The output of this unit is the final output of the error correction system and includes a flag indicating if the computation was in error. An illustration of ECU is shown in Fig. 6.

6.2 *Identifying Robust Neural Architectures*

Some DNN topologies (neural architectures) are more robust than others against device variations. Finding these neural architectures is a viable way of mitigating the effect of device variations. Meanwhile, different neural architectures require different amounts of computation power and are thus with different inference latency and power consumption. Handcrafting a neural architecture that meets all design requirements is a challenging task. Fortunately, neural architecture search (NAS) [48, 52, 57] is proposed to automatically find an optimal neural architecture in a designated design space using reinforcement learning-based algorithms.

In this section, we introduce NACIM [23], a device-circuit-architecture co-exploration framework that can automatically identify the best CiM neural accelerators from a design space including the device type, circuit topology, and neural architecture hyper-parameters. NACIM framework iteratively conducts explorations based on a reward function, which is suitable for reinforcement learning approaches

or evolutionary algorithms. By configuring the parameters of the framework, designers can customize the optimization goals in terms of their demands. The authors model the effect of device variation by modeling the shot noise as a stuck-at-low or stuck-at-high fault, and the other noise sources as a whole to be a zero-mean additional Gaussian noise extracted from widely adopted models [56] on the weight value. Experimental results show that the proposed NACIM framework can find the robust neural network with only 0.45% accuracy loss in the presence of device variation, compare with a 76.44% loss from the state-of-the-art NAS without considering device variation.

6.3 Training Robust DNNs

DNN models with the same neural architecture but different weights can have very similar accuracy in ideal conditions but very different accuracy in the existence of device variations. Thus, finding proper weights that are robust against device variations in the training process is a desirable approach.

A straightforward way to find robust weights is to simulate the noisy forward path in the training process, i.e., in each iteration of training, the algorithm sample an instance of noise and add it to the weight in the forward and backpropagation path to calculate the gradient, then remove the noise when updating the weights.

This method is used in NACIM [23] which is introduced before. For implementations in MNIST dataset [27], noise injection training can reduce the accuracy drop between the ideal model and model with device variations from 6 to 0.5%, and in CIFAR-10 dataset [25], noise injection training can reduce the accuracy drop from 76.44 to 0.45%.

A more advanced way to find robust weights is to seek help from Bayesian Neural Networks (BNN). Bayesian neural network is known for a stochastic gradient variational Bayes framework applied to approximate posterior distributions over network parameters. By employing a prior distribution over the weight space, BNN allows us to introduce variation to the learning process to better fit the observations [15].

A recent work [15] uses BNN to improve the robustness of nvCiM accelerators. BNN requires a priori distribution and uses an estimated posterior to fit this distribution. The priori can be obtained from device variation models. These models are inferred from expert knowledge with the help of measurement, simulation, and historical data. The authors also use KL divergence as the regularization term to enforce the memristor variation structural characteristics.

Although the priori used in most recent works are carefully designed, they can still be imprecise or uncertain because of the measurement imperfectness and the ever-going evolution of emerging devices. To address this issue, the authors of [15] propose a variance-adaptive priori to weigh the value of prior knowledge. The author modify the optimization objective of BNNs so that weights with larger values are more regularized by the priori, i.e., it allows placing heavier priorities on those critical weights (with higher magnitude) on crossbar arrays that are prone to receive

more impact from device variations, thereby reducing oscillations in convergence for more efficient training. Finally, to prevent the over-amplification of variation during training, the authors add an additional regularization term using L2 norm loss. In CIFAR-10 dataset, this proposed method is able to reduce the accuracy drop from 45.7 to 0.3%.

Although these two methods are effective in terms of mitigating the effect of device variations on nvCiM accelerators, they require much more training iterations to converge compared with traditional training methods. In the MNIST dataset, both methods require at least 10× more iterations of training to reach a similar accuracy as the traditional training method [23].

7 Conclusions

Computing-in-memory with emerging non-volatile devices (nvCiM) is a great candidate for efficient DNN acceleration because of its unique architecture that breaks the memory wall. However, it suffers from unreliability issues, especially the device variation issues of emerging NV devices. Understanding the property of emerging NV devices and the general architecture of nvCiM DNN accelerators helps to better model the effect of device unreliability circuit and application level. The modeling of unreliability also helps in mitigating the impact of device variations. The representative ways of mitigation include the adoption of ECC in the architecture and finding neural network topologies and training DNN weights that are more robust against device variations.

References

1. Alibart F, Gao L, Hoskins BD, Strukov DB (2012) High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. Nanotechnology 23(7):075201
2. Aziz A, Breyer ET, Chen A, Chen X, Datta S, Gupta SK, Hoffmann M, Hu XS, Ionescu A, Jerry M et al (2018) Computing with ferroelectric FETs: devices, models, systems, and applications. In: 2018 Design, automation & test in Europe conference & exhibition (DATE). IEEE, pp 1289–1298
3. Aziz A, Ghosh S, Datta S, Gupta SK (2016) Physics-based circuit-compatible spice model for ferroelectric transistors. IEEE Electron Device Lett 37(6):805–808
4. Beck A, Bednorz J, Gerber C, Rossel C, Widmer D (2000) Reproducible switching effect in thin oxide films for memory applications. Appl Phys Lett 77(1):139–141
5. Berger L (1996) Emission of spin waves by a magnetic multilayer traversed by a current. Phys Rev B 54(13):9353
6. Carboni R, Ambrogio S, Chen W, Siddik M, Harms J, Lyle A, Kula W, Sandhu G, Ielmini D (2016) Understanding cycling endurance in perpendicular spin-transfer torque (p-STT) magnetic memory. In: 2016 IEEE International electron devices meeting (IEDM). IEEE, pp 21–6
7. Chappert C, Fert A, Van Dau FN (2010) The emergence of spin electronics in data storage. Nanosci Technol Collect Rev Nat J 147–157

8. Chen WH, Dou C, Li KX, Lin WY, Li PY, Huang JH, Wang JH, Wei WC, Xue CX, Chiu YC et al (2019) CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. Nat Electron 2(9):420–428
9. Chen Y, Luo T, Liu S, Zhang S, He L, Wang J, Li L, Chen T, Xu Z, Sun N et al (2014) DaDianNao: a machine-learning supercomputer. In: 2014 47th Annual IEEE/ACM international symposium on microarchitecture. IEEE, pp 609–622
10. Chi P, Li S, Xu C, Zhang T, Zhao J, Liu Y, Wang Y, Xie Y (2016) PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. ACM SIGARCH Comput Architect News 44(3):27–39
11. Choi BJ, Torrezan AC, Strachan JP, Kotula P, Lohn A, Marinella MJ, Li Z, Williams RS, Yang JJ (2016) High-speed and low-energy nitride memristors. Adv Funct Mater 26(29):5290–5296
12. Draper J, Chame J, Hall M, Steele C, Barrett T, LaCoss J, Granacki J, Shin J, Chen C, Kang CW et al (2002) The architecture of the diva processing-in-memory chip. In: Proceedings of the 16th international conference on supercomputing, pp 14–25
13. Farmahini-Farahani A, Ahn JH, Morrow K, Kim NS (2015) NDA: near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In: 2015 IEEE 21st International symposium on high performance computer architecture (HPCA). IEEE, pp 283–295
14. Feinberg B, Wang S, Ipek E (2018) Making memristive neural network accelerators reliable. In: 2018 IEEE International symposium on high performance computer architecture (HPCA). IEEE, pp 52–65
15. Gao D, Huang Q, Zhang L, Yin X, Li B, Schlichtmann U, Zhuo C (2021) Bayesian inference based robust computing on memristor crossbar. In: 2021 56th ACM/IEEE Design automation conference (DAC). IEEE, pp 1–6
16. Gao D, Reis D, Hu XS, Zhuo C (2019) Eva-CiM: a system-level energy evaluation framework for computing-in-memory architectures. arXiv preprint arXiv:1901.09348
17. Han S, Mao H, Dally WJ (2015) Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint arXiv:1510.00149
18. Hsieh K, Khan S, Vijaykumar N, Chang KK, Boroumand A, Ghose S, Mutlu O (2016) Accelerating pointer chasing in 3d-stacked memory: challenges, mechanisms, evaluation. In: 2016 IEEE 34th International conference on computer design (ICCD). IEEE, pp 25–32
19. Ielmini D (2011) Modeling the universal set/reset characteristics of bipolar RRAM by field-and temperature-driven filament growth. IEEE Trans Electron Devices 58(12):4309–4317
20. Ielmini D, Nardi F, Cagli C (2010) Resistance-dependent amplitude of random telegraph-signal noise in resistive switching memories. Appl Phys Lett 96(5):053503
21. Ielmini D, Wong HSP (2018) In-memory computing with resistive switching devices. Nat Electron 1(6):333–343
22. Jain S, Ranjan A, Roy K, Raghunathan A (2017) Computing in memory with spin-transfer torque magnetic ram. IEEE Trans Very Large Scale Integr (VLSI) Syst 26(3):470–483
23. Jiang W, Lou Q, Yan Z, Yang L, Hu J, Hu XS, Shi Y (2020) Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. IEEE Trans Comput
24. Kim KM, Jeong DS, Hwang CS (2011) Nanofilamentary resistive switching in binary oxide system; a review on the present status and outlook. Nanotechnology 22(25):254002
25. Krizhevsky A et al (2009) Learning multiple layers of features from tiny images
26. Kvatinsky S, Belousov D, Liman S, Satat G, Wald N, Friedman EG, Kolodny A, Weiser UC (2014) Magic—memristor-aided logic. IEEE Trans Circ Syst II Express Briefs 61(11):895–899
27. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
28. Lee MJ, Lee CB, Lee D, Lee SR, Chang M, Hur JH, Kim YB, Kim CJ, Seo DH, Seo S et al (2011) A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta_2O_{5-x}/TaO_{2-x} bilayer structures. Nat Mater 10(8):625–630
29. Li KS, Chen PG, Lai TY, Lin CH, Cheng CC, Chen CC, Wei YJ, Hou YF, Liao MH, Lee MH et al (2015) Sub-60 mV-swing negative-capacitance FinFET without hysteresis. In: 2015 IEEE International electron devices meeting (IEDM). IEEE, pp 22–6

30. Li S, Xu C, Zou Q, Zhao J, Lu Y, Xie Y (2016) Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In: Proceedings of the 53rd annual design automation conference, pp 1–6
31. Li X, Sampson J, Khan A, Ma K, George S, Aziz A, Gupta SK, Salahuddin S, Chang MF, Datta S et al (2017) Enabling energy-efficient nonvolatile computing with negative capacitance FET. IEEE Trans Electron Devices 64(8):3452–3458
32. Liu Q, Sun J, Lv H, Long S, Yin K, Wan N, Li Y, Sun L, Liu M (2012) Real-time observation on dynamic growth/dissolution of conductive filaments in oxide-electrolyte-based ReRAM. Adv Mater 24(14):1844–1849
33. Locatelli N, Cros V, Grollier J (2014) Spin-torque building blocks. Nat Mater 13(1):11–20
34. Loke D, Lee T, Wang W, Shi L, Zhao R, Yeo Y, Chong T, Elliott S (2012) Breaking the speed limits of phase-change memory. Science 336(6088):1566–1569
35. Mai K, Paaske T, Jayasena N, Ho R., Dally WJ, Horowitz M (2000) Smart memories: a modular reconfigurable architecture. In: Proceedings of 27th international symposium on computer architecture (IEEE Cat. No. RS00201). IEEE, pp 161–171
36. Niu D, Xiao Y, Xie Y (2012) Low power memristor-based ReRAM design with error correcting code. In: 17th Asia and South Pacific design automation conference. IEEE, pp 79–84
37. Oskin M, Chong FT, Sherwood T (1998) Active pages: a computation model for intelligent memory. In: Proceedings of the 25th annual international symposium on computer architecture (Cat. No. 98CB36235). IEEE, pp 192–203
38. Reis D, Niemier M, Hu XS (2018) Computing in memory with FeFETs. In: Proceedings of the international symposium on low power electronics and design, pp 1–6
39. Shafiee A, Nag A, Muralimanohar N, Balasubramonian R, Strachan JP, Hu M, Williams RS, Srikumar V (2016) Isaac: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. ACM SIGARCH Comput Architect News 44(3):14–26
40. Sharma P, Tapily K, Saha A, Zhang J, Shaughnessy A, Aziz A, Snider G, Gupta S, Clark R, Datta S (2017) Impact of total and partial dipole switching on the switching slope of gate-last negative capacitance FETs with ferroelectric hafnium zirconium oxide gate stack. In: 2017 Symposium on VLSI technology. IEEE, pp T154–T155
41. Slonczewski JC (1996) Current-driven excitation of magnetic multilayers. J Magn Magn Mater 159(1–2):L1–L7
42. Sze V, Chen YH, Yang TJ, Emer JS (2017) Efficient processing of deep neural networks: a tutorial and survey. Proc IEEE 105(12):2295–2329
43. Van Lint J, van der Geer G (2012) Introduction to coding theory and algebraic geometry, vol 12. Birkhäuser
44. Wang D, George S, Aziz A, Datta S, Narayanan V, Gupta SK (2016) Ferroelectric transistor based non-volatile flip-flop. In: Proceedings of the 2016 international symposium on low power electronics and design, pp 10–15
45. Wang K, Liu Z, Lin Y, Lin J, Han S (2019) HAQ: hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 8612–8620
46. Xia L, Liu M, Ning X, Chakrabarty K, Wang Y (2017) Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems. In: Proceedings of the 54th annual design automation conference 2017, pp 1–6
47. Xu C, Niu D, Muralimanohar N, Jouppi NP, Xie Y (2013) Understanding the trade-offs in multi-level cell ReRAM memory design. In: 2013 50th ACM/EDAC/IEEE Design automation conference (DAC). IEEE, pp 1–6
48. Yan Z, Jiang W, Hu XS, Shi Y (2021) Radars: memory efficient reinforcement learning aided differentiable neural architecture search. arXiv preprint arXiv:2109.05691
49. Yan Z, Juan DC, Hu XS, Shi Y (2021) Uncertainty modeling of emerging device based computing-in-memory neural accelerators with application to neural architecture search. In: 2021 26th Asia and South Pacific design automation conference (ASP-DAC). IEEE, pp 859–864
50. Yan Z, Shi Y, Liao W, Hashimoto M, Zhou X, Zhuo C (2020) When single event upset meets deep neural networks: observations, explorations, and remedies. In: 2020 25th Asia and South Pacific design automation conference (ASP-DAC). IEEE, pp 163–168

51. Yang JJ, Strukov DB, Stewart DR (2013) Memristive devices for computing. Nat Nanotechnol 8(1):13–24
52. Yang L, Yan Z, Li M, Kwon H, Lai L, Krishna T, Chandra V, Jiang W, Shi Y (2020) Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks. In: 2020 57th ACM/IEEE Design automation conference (DAC). IEEE, pp 1–6
53. Yuasa S, Nagahama T, Fukushima A, Suzuki Y, Ando K (2004) Giant room-temperature magnetoresistance in single-crystal Fe/MgO/Fe magnetic tunnel junctions. Nat Mater 3(12):868–871
54. Zaman KS, Reaz MBI, Ali SHM, Bakar AAA, Chowdhury MEH (2021) Custom hardware architectures for deep learning on portable devices: a review. IEEE Trans Neural Networks Learn Syst
55. Zhang D, Jayasena N, Lyashevsky A, Greathouse JL, Xu L, Ignatowski M (2014) TOP-PIM: throughput-oriented programmable processing in memory. In: Proceedings of the 23rd international symposium on high-performance parallel and distributed computing, pp 85–98
56. Zhao M, Wu H, Gao B, Zhang Q, Wu W, Wang S, Xi Y, Wu D, Deng N, Yu S et al (2017) Investigation of statistical retention of filamentary analog RRAM for neuromorphic computing. In: 2017 IEEE International electron devices meeting (IEDM). IEEE, pp 39–4
57. Zoph B, Le QV (2017) Neural architecture search with reinforcement learning. In: International conference on learning representations (ICLR)