Mapping-aware Biased Training for Accurate Memristor-based Neural Networks

Diware, Sumit; Gebregiorgis, Anteneh; Joshi, Rajiv V.; Hamdioui, Said; Bishnoi, Rajendra

DOI
10.1109/AICAS57966.2023.10168661

Publication date
2023

Document Version
Final published version

Published in
AICAS 2023 - IEEE International Conference on Artificial Intelligence Circuits and Systems, Proceeding

2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS) | 979-8-3503-3267-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/AICAS57966.2023.10168661

Mapping-aware Biased Training for Accurate Memristor-based Neural Networks

Sumit Diware* Anteneh Gebregiorgis* Rajiv V. Joshi† Said Hamdioui* Rajendra Bishnoi*
*Computer Engineering Lab, Delft University of Technology, Delft, The Netherlands.
Email: {S.S.Diware, A.B.Gebregiorgis, S.Hamdioui, R.K.Bishnoi}@tudelft.nl
†IBM Research Division, Yorktown Heights, NY, USA. Email: rvjoshi@us.ibm.com

***Abstract*—Memristor-based computation-in-memory (CIM) can achieve high energy efficiency by processing the data within the memory, which makes it well-suited for applications like neural networks. However, memristors suffer from conductance variation problem where their programmed conductance values deviate from the desired values. Such variations lead to computational errors that result in degraded inference accuracy in CIM-based neural networks. In this paper, we present a mapping-aware biased training methodology to mitigate the impact of conductance variation on CIM-based neural networks. We first determine which conductance states of the memristor are inherently more immune to variation. The neural network is then trained under the constraint that important weights can only take numeric values which directly get mapped to such favorable states. Simulation results show that our proposed mapping-aware biased training achieves up to 2.4× hardware accuracy compared to the conventional training.**

I. Introduction

Neural networks perform various cognitive tasks without explicit programming, making them the heart of modern artificial intelligence (AI) [1]. These networks are conventionally deployed on von-Neumann architecture-based hardware like CPUs, GPUs, and ASICs for AI such as TPUs [2]–[4]. The physical separation of memory and computing units in such hardware results in low energy efficiency due to memory wall [5]–[7]. *Computation-in-memory* (CIM) can overcome this problem by performing computations within the memory [8]–[10]. It uses emerging non-volatile memory technologies such as memristors, also called resistive random access memories (RRAMs), which are highly scalable and compatible with CMOS technology [11]. However, memristors suffer from conductance variation problem where their programmed conductance deviates from the target value, due to fabrication imperfections and stochastic device physics [12]. This leads to an undesired change in the neural network weights stored as memristor conductances, resulting in low accuracy.

Prior works addressing the conductance variation issue in CIM-based neural networks can be grouped into four categories: i) on-chip training, ii) off-chip training or mapping based on hardware characterization, iii) hardware compensation, and iv) write-verify programming. First, on-chip training inherently adapts the weights to conductance variation as the network is trained on the CIM chip [13], [14]. However, it is not scalable due to individual training necessity for each chip, high energy consumption, and endurance issues. Second, off-chip training using a hardware-calibrated software model of conductance variation [15], [16] is also not scalable, as each chip requires individual characterization and training. Moreover, some works [17], [18] prevent large weights from mapping to high variation memristors. This requires extensive chip characterization and does not address errors due to the accumulation of variations in small weights. Alternatively, noise estimated from a non-extensive chip characterization can be injected in off-chip training [19]–[24] to enhance the network's tolerance towards errors due to conductance variation. However, this approach fails to address the issue of reducing such errors, as memristors can still get mapped to high variation conductance states, rendering it ineffective. Last, the hardware compensation and write-verify programming involve significant energy and area overheads with increased design complexity [25]–[28]. Hence, there is a strong need for an effective, scalable, and low-overhead solution to mitigate conductance variation impact on CIM-based neural networks.

In this paper, we present a mapping-aware biased training methodology to improve the accuracy of CIM-based neural networks in the presence of conductance variation. We first identify memristor conductance states with low variation impact (favorable states). We then derive a favorability constraint that only allows weight values that map to these favorable states. During training, we determine which weights are important for CIM hardware accuracy and impose the favorability constraint on them. The resulting post-training values of these important weights then directly map to favorable states, leading to high inference accuracy on CIM hardware. The key contributions of this paper can be summarized as follows:

- A favorability constraint analysis to find the weight values desirable for reducing conductance variation errors.
- An approach to identify the important weights which significantly influence the hardware accuracy.
- A mapping-aware biased training with favorability constraint on important weights for high hardware accuracy.

The proposed training methodology is effective and scalable,

This work was supported in part by the EU H2020 grant "DAIS" that has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No. 101007273.

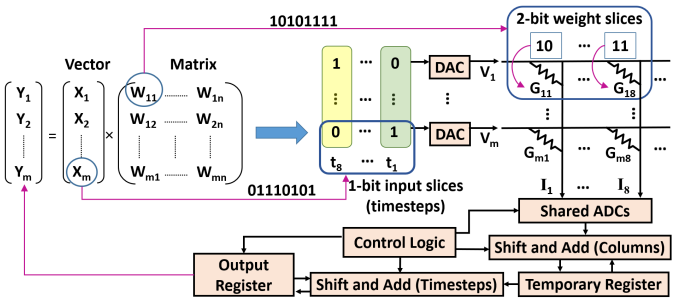


Fig. 1. Vector-matrix multiplication using memristor-based CIM.

as it does not require extensive chip characterization and relies only on the order of conductance states based on their variation impact. It achieves up to 2.4× hardware accuracy compared to conventional training, without any hardware overhead.

The rest of this paper is organized as follows: Section II presents the basics of CIM. The details of the proposed training methodology are described in Section III, followed by simulation results in Section IV and conclusion in Section V.

II. Background

A. Computation-In-Memory Architecture for Neural Networks

Computation-in-memory (CIM) hardware provides energy-efficient inference for software-trained neural networks. It achieves this by in-situ vector-matrix multiplication as shown in Fig. 1. The full-precision weights and inputs are split into smaller slices as i) bit-capacity of memristors is insufficient for weights and ii) full-precision inputs need digital-to-analog converters (DACs) and analog-to-digital converters (ADCs) which consume huge energy and area [29]. Weight slices are mapped to memristor conductances (G's) and input slices are converted to voltages (V's) by DACs. The ADCs then convert output currents (I's) to digital values, which undergo shift-and-add post-processing to obtain the full-precision output.

B. Memristor Technology

Memristor, also called resistive random access memory (RRAM), typically consists of an oxide layer sandwiched between two metal electrodes as shown in Fig 2. It stores data in the form of conductance. "SET" process creates oxygen vacancies in the oxide to increase its conductance, while "RESET" process depletes the oxygen vacancies to reduce its conductance. As shown in Fig 2, a single memristor can store multiple bits by partial SET/RESET processes [30]. To read the data (conductance) stored in the memristor, a small voltage is applied across it and the resulting current is measured.

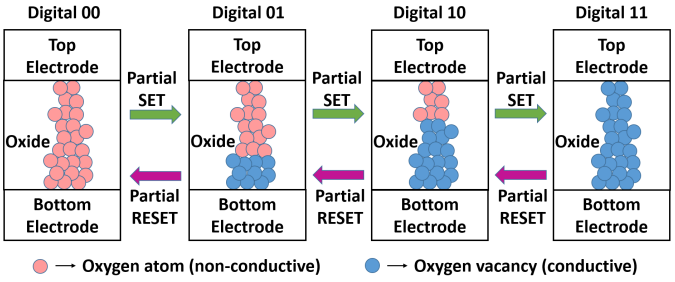
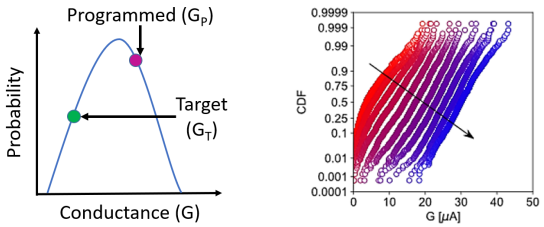


Fig. 2. Operation of memristor device storing two bits.



(a) Concept illustration. (b) Measurements in [31].

Fig. 3. Conductance variation in memristors.

C. Conductance Variation

The programmed conductance of a memristor deviates from its target value due to the stochastic nature of oxygen vacancy creation/depletion and fabrication imperfections like variable oxide thickness [18]. This phenomenon is called conductance variation, shown in Fig. 3. It leads to incorrect weight storage as memristor conductance, resulting in poor accuracy. In this paper, we improve the accuracy of memristor-based neural network architectures in the presence of conductance variation.

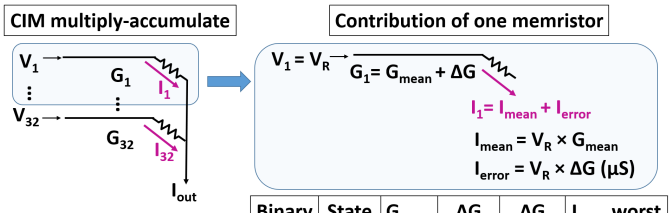
III. Proposed Mapping-Aware Biased Training

A. Favorable Conductance States Analysis

Fig. 4 shows a CIM-based multiply-accumulate operation, where I_{error} is the error current in a single memristor device due to conductance variation. As small I_{error} is desirable, the preference order of states in Fig. 4 is: G_{00} (best), G_{01}, G_{11}, G_{10} (worst). Despite having a higher variation percentage, G_{00} and G_{01} are preferred over G_{11} and G_{10} as their small mean values result in small I_{error}. Hence, the preference order of conductance states must be based on I_{error} contribution instead of the variation percentage. The ordered conductance states are then grouped into: i) unfavorable states (U) to avoid for mapping, and ii) favorable states (F) to prefer for mapping. Based on Fig. 4, the possible grouping configurations are:

- Config-1: F={G_{00}}, U={G_{01},G_{11},G_{10}}
- Config-2: F={G_{00}, G_{01}}, U={G_{11},G_{10}}
- Config-3: F={G_{00}, G_{01}, G_{11}}, U={G_{10}}

Config-1 sets weights only to zero while config-2 forces them to the same sign. This is undesirable as the neural network requires both positive and negative non-zero weights. As config-3 can represent non-zero weights with different signs, it is used in our mapping-aware biased training methodology.



V_R : Input voltage (0.4 V)

G_{mean} : Mean conductance

ΔG : Worst case variation

Binary value	State name	G_{mean} (μS)	ΔG (%)	ΔG (μS)	I_{error} worst case (μA)
00	G_{00}	2	38.5	0.8	1.6
01	G_{01}	67	24.8	16.6	6.6
10	G_{10}	134	17.2	23.1	9.2
11	G_{11}	200	9.3	18.6	7.4

Fig. 4. Favorable conductance states analysis for a 2-bit memristor (four conductance states). The used conductance variation data is obtained from [32].

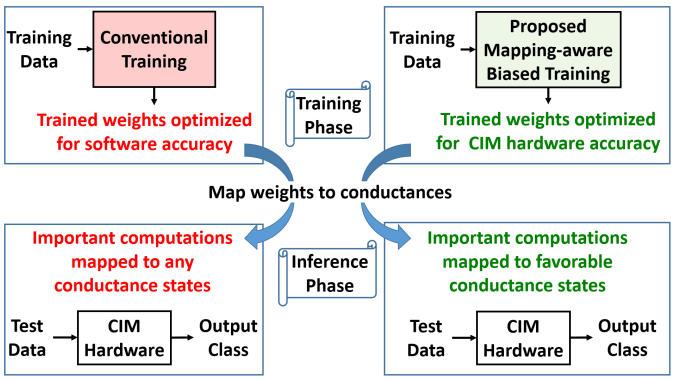


Fig. 5. Overview of the conventional and proposed training methodologies.

B. Mapping-aware Biased Training Methodology

1) Overview: The deployment of a neural network on CIM hardware for inference involves two phases as shown in Fig. 5: i) Training the neural network weights to obtain high classification accuracy. ii) Mapping the trained weights to memristor conductances for inference on CIM hardware. Conventional training can result in the mapping of weights to conductance states having a high variation impact (unfavorable states). This can lead to low hardware accuracy despite high software accuracy. Our proposed mapping-aware biased training restricts the neural network weights during training, so that their post-training values directly get mapped to conductance states having a low variation impact (favorable states). However, restricting too many weights hinders backpropagation and leads to low software accuracy. This in turn results in low hardware accuracy, as it is upper bounded by software accuracy. Conversely, if too few weights are restricted, the hardware accuracy will be poor as many memristors can get mapped to unfavorable states. Hence, our proposed mapping-aware biased training only restricts the important weights. This leads to high software accuracy due to the adaptability of non-important weights and also provides high hardware accuracy

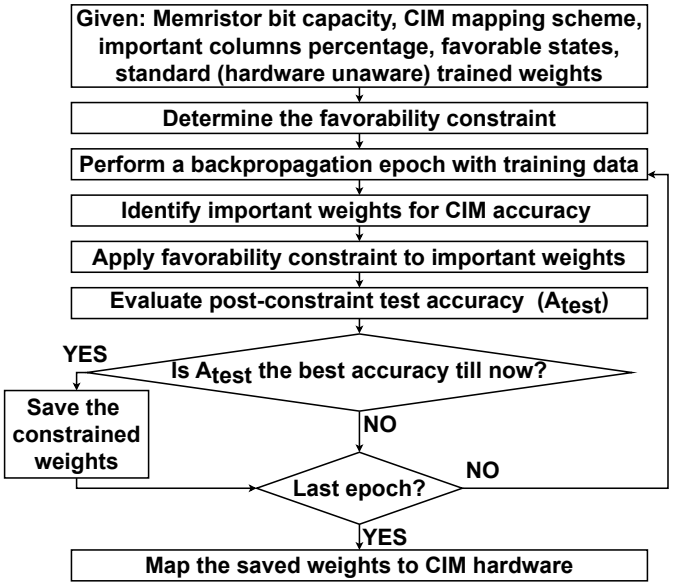


Fig. 6. Flowchart of the proposed mapping-aware biased training.

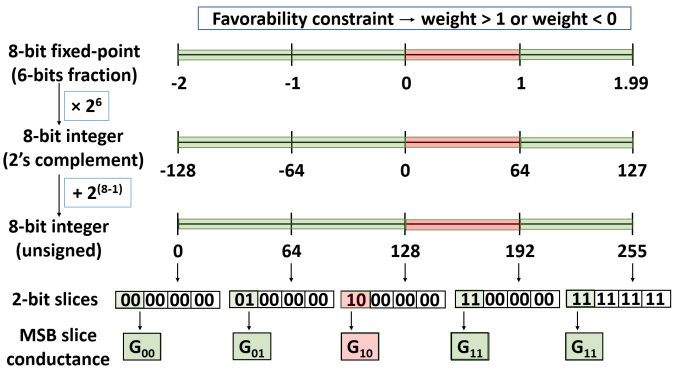


Fig. 7. Illustration of favorability constraint derivation for mapping MSB slice of 8-bit weight to favorable conductance states in a 2-bit memristor.

as important weights get mapped to favorable memristor states.

2) Biased Training: The flowchart of our mapping-aware biased training is shown in Fig. 6. We first train the neural network in a standard (hardware-unaware) manner. These weights are used as initial weights for mapping-aware biased training for faster convergence. We then determine a favorability constraint on the weights to ensure the mapping of desired weight bits to favorable conductance states. It depends on memristor bit capacity and CIM mapping scheme details like fixed-point format, underlying CIM architecture, etc. For example, consider 2-bit memristors (slices), 8-bit fixed-point weights (6-bit fraction), and CIM architecture in [29]. The mapping scheme first converts trained weights to 2's complement fixed-point format. It then shifts the 2's complement weight range by 2^7 to overcome the difficulty in isolating the sign contribution from a multi-bit slice [29]. Fig. 7 then shows the favorability constraint to map the most significant 2-bit slice to favorable states in Section III-A (G_{00}, G_{01}, and G_{10}) for this example.

We now perform a new epoch of backpropagation using training data and then determine which weights are important for high hardware accuracy. In a neural network, some weights have more importance than others for high software accuracy. However, in CIM hardware design for the same network, instead of individual weights, some crossbar columns (groups of weights) are more important than others for high hardware accuracy. This is because the basic computation in CIM is the column-wise multiply-accumulate operation. Let HI_c denote the importance of a CIM column for high hardware accuracy and SI_w denote the importance of a weight for high software accuracy. If P denotes the network output (without softmax) and L denotes the one hot label, then SI_w is given by Eq. 1.

$$\text{SI}_w = \frac{\partial Q}{\partial w}\ , \text{where } Q = \sum_{i=1}^{\text{Batch size}} P_i \times L_i \qquad (1)$$

HI_c is then obtained by dividing the software weight matrix into crossbar-sized chunks and adding SI_w of weights per column across all chunks. A high HI_c value indicates more influence on hardware accuracy. We now select $m\%$ columns with the highest HI_c (m is obtained by design-space exploration, details in Section IV-B). Weights in these columns are restricted as per the favorability constraint and test accuracy is evaluated. This process is repeated for a given number

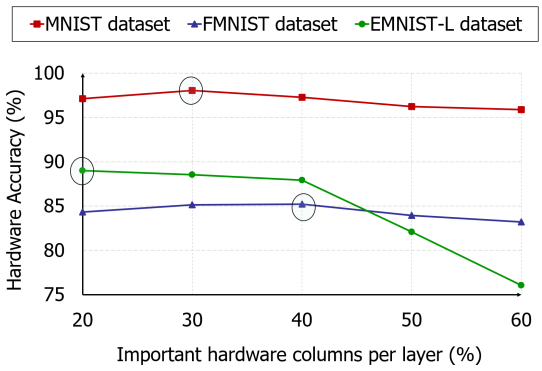


Fig. 8. Design-space exploration for the percentage of important columns per neural network layer. Circle denotes the peak hardware accuracy per dataset.

of biased training epochs and weights with the best post-restriction test accuracy are mapped to CIM hardware.

IV. Simulation Results

A. Simulation Setup

We have developed a Python-based framework for behavioral simulation of neural network inference on CIM hardware. It is based on in-situ multiply-accumulate (IMA) unit in state-of-the-art CIM architectures [29], [33]. Power and area for various IMA components are also obtained from [29]. We consider 8-bit weights split across four memristors of 2-bit capacity. Memristor device parameters and conductance variation data are obtained from [32] which presents experiments on real RRAM devices. We have performed evaluations using MNIST [34], Fashion MNIST (FMNIST) [35], and EMNIST letters (EMNIST-L) [36] datasets on LeNet-5 neural network [34]. The mapping-aware biased training is carried out in software and trained weights are used in our Python-based framework to evaluate the hardware inference accuracy.

B. Neural Network Accuracy

1) Design-space Exploration: We perform design space exploration to determine the optimal percentage of crossbar columns, which are designated as important and subjected to favorability constraints in all neural network layers, as depicted in Fig. 8. A high percentage results in low software accuracy as restricting more weights obstructs the minimization of the cost function. The hardware accuracy is also reduced as it is upper bounded by software accuracy. A moderate

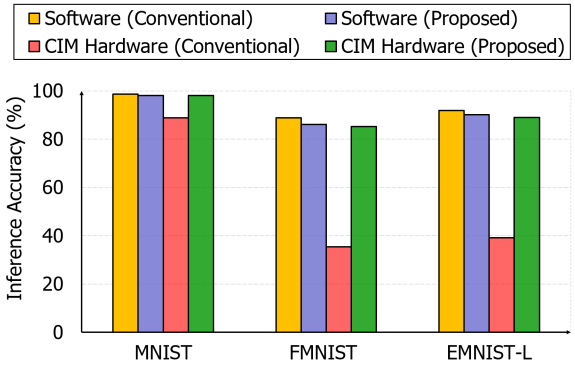


Fig. 9. Neural network inference accuracy comparison across various datasets.

TABLE I
Hardware metrics per in-situ multiply-accumulate unit.

Metric	Conventional Training	Proposed Mapping-aware Biased Training
FMNIST accuracy (%)	35.4	85.2
Energy consumption (pJ)	3738	3738
Area (μm^2)	21765	21765
Correct operations per unit energy for FMNIST (GOP/J)	96.9	233.4

percentage can provide high accuracy in both software and hardware by balancing the freedom of non-important weights and restriction on important weights. The optimal percentage varies from one dataset to another as indicated in Fig. 8.

2) Accuracy Comparison: The accuracy comparison between proposed mapping-aware biased training and conventional training (backpropagation) is shown in Fig. 9. The proposed mapping-aware biased training has a slightly lower software accuracy compared to conventional training. This is because cost function minimization during training becomes difficult due to the favorability constraint on important weights. Our proposed biased training provides up to 2.4× hardware accuracy compared to conventional training. This can be attributed to the mapping of important weights to conductance states having a low variation impact. The accuracy improvement is higher for complex datasets (FMNIST, EMNIST-L) than simpler ones (MNIST), as they need more error-free computations for correct classification.

C. Hardware Metrics

The comparison of hardware metrics between the proposed mapping-aware biased training and the conventional training (backpropagation) is shown in Table I. They both need identical hardware components and hence consume the same energy and area. We define a new metric "correct operations per unit energy" as the ratio of the number of correct operations to energy consumption (unit: Giga-operations per joule (GOP/J)). Here, the number of correct operations is the product of accuracy (as fraction) and the total number of operations. Table I shows that the proposed mapping-aware biased training achieves up to 2.4× correct operations per unit energy than conventional training without any hardware overhead.

V. Conclusion

We have presented a mapping-aware biased training to mitigate the impact of conductance variation on CIM-based neural networks. This was achieved by restricting the important weights during training, so that their post-training values directly get mapped to conductance states with low variation impact. The proposed biased training achieves up to 2.4× hardware accuracy and up to 2.4× correct operations per unit energy compared to the conventional training, without incurring any hardware overhead. Such high accuracy and energy efficiency can facilitate the deployment of CIM-based neural networks for edge-AI.

REFERENCES

[1] C. Szegedy *et al.*, "Going deeper with convolutions," in *CVPR*, 2015.
[2] Intel Processor Family. [Online]. Available: https://www.intel.in/content/www/in/en/products/processors/core.html
[3] NVIDIA Turing Architecture GPUs. [Online]. Available: https://www.nvidia.com/en-in/geforce/turing/
[4] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.
[5] F. Cai *et al.*, "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations," *Nature Electronics*, 2019.
[6] S. Diware *et al.*, "Accurate and energy-efficient bit-slicing for rram-based neural networks," *TETCI*, 2023.
[7] A. Singh *et al.*, "Srif: Scalable and reliable integrate and fire circuit adc for memristor-based cim architectures," *TCAS-I*, 2021.
[8] P.-Y. Chen *et al.*, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," *TCAD*, 2018.
[9] M. Zahedi *et al.*, "System design for computation-in-memory: From primitive to complex functions," in *VLSI-SoC*, 2022.
[10] T. Shahroodi *et al.*, "Demeter: A fast and energy-efficient food profiler using hyperdimensional computing in memory," *IEEE Access*, 2022.
[11] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, 2020.
[12] P.-Y. Chen and S. Yu, "Technological Benchmark of Analog Synaptic Devices for Neuroinspired Architectures," *IDT*, 2019.
[13] S. R. Nandakumar *et al.*, "Mixed-precision deep learning based on computational memory," *Frontiers in Neuroscience*, 2020.
[14] C. Li *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature Communications*, 2018.
[15] G. Charan *et al.*, "Accurate inference with inaccurate rram devices: A joint algorithm-design solution," *JXCDC*, 2020.
[16] W. Jiang *et al.*, "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *TC*, 2021.
[17] Z. Song *et al.*, "ITT-RNA: Imperfection tolerable training for rram-crossbar-based deep neural-network accelerator," *TCAD*, 2021.
[18] L. Chen *et al.*, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Design, Automation & Test in Europe Conference Exhibition (DATE)*, 2017, pp. 19–24.
[19] A. Antolini *et al.*, "Combined hw/sw drift and variability mitigation for pcm-based analog in-memory computing for neural network applications," *JESTCS*, 2023.
[20] J. Doevenspeck *et al.*, "Oxrram-based analog in-memory computing for deep neural network inference: A conductance variability study," *TED*, 2021.
[21] M. Fritscher *et al.*, "Mitigating the effects of rram process variation on the accuracy of artificial neural networks," in *SAMOS*, 2021.
[22] C. Huang *et al.*, "An efficient variation-tolerant method for rram-based neural network," in *ICET*, 2022.
[23] V. Joshi *et al.*, "Accurate deep neural network inference using computational phase-change memory," *Nature Communications*, 2020.
[24] Q. Wang, Y. Park, and W. D. Lu, "Device variation effects on neural network inference accuracy in analog in-memory computing systems," *AIS*, 2022.
[25] V. Milo *et al.*, "Optimized programming algorithms for multilevel rram in hardware neural networks," in *IRPS*, 2021.
[26] N. Lepri *et al.*, "Mitigating read-program variation and ir drop by circuit architecture in rram-based neural network accelerators," in *IRPS*, 2022.
[27] J. He *et al.*, "Rvcomp: Analog variation compensation for rram-based in-memory computing," in *ASPDAC*, 2023.
[28] C.-C. Chang *et al.*, "Device quantization policy in variation-aware in-memory computing design," *Nature Scientific Reports*, 2022.
[29] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.
[30] W. Kim *et al.*, "Multistate memristive tantalum oxide devices for ternary arithmetic," *Nature Scientific reports, 2016*, 2016.
[31] G. Pedretti *et al.*, "Conductance variations and their impact on the precision of in-memory computing with resistive switching memory (RRAM)," in *IRPS*, 2021.
[32] A. Prakash and H. Hwang, "Multilevel Cell Storage and Resistance Variability in Resistive Random Access Memory," *PSR*, 2016.
[33] A. Ankit *et al.*, "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference," in *ASPLOS*, 2019.
[34] Y. Lecun *et al.*, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, 1998.
[35] H. Xiao *et al.*, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv*, 2017.
[36] G. Cohen *et al.*, "EMNIST: Extending MNIST to handwritten letters," in *IJCNN*, 2017.