Dynamic Detection and Mitigation of Read-disturb for Accurate Memristor-based Neural Networks

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

DOI
10.1109/aicas59952.2024.10595966

Publication date
2024

Document Version
Final published version

Published in
2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)

Dynamic Detection and Mitigation of Read-disturb for Accurate Memristor-based Neural Networks

Sumit Diware* Mohammad Amin Yaldagard* Anteneh Gebregiorgis* Rajiv V. Joshi†
Said Hamdioui* Rajendra Bishnoi*

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

2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS) | 979-8-3503-8363-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/AICAS59952.2024.10595966

***Abstract*—Computation-in-memory (CIM) using memristors can facilitate data processing within the memory itself, leading to superior energy efficiency than conventional von-Neumann architecture. This makes CIM well-suited for data-intensive applications like neural networks. However, a large number of read operations can induce an undesired resistance change in the memristor, known as read-disturb. As memristor resistances represent the neural network weights in CIM hardware, read-disturb causes an unintended change in the network's weights that leads to poor accuracy. In this paper, we propose a methodology for read-disturb detection and mitigation in CIM-based neural networks. We first analyze the key insights regarding the read-disturb phenomenon. We then introduce a mechanism to dynamically detect the occurrence of read-disturb in CIM-based neural networks. In response to such detections, we develop a method that adapts the sensing conditions of CIM hardware to provide error-free operation even in the presence of read-disturb. Simulation results show that our proposed methodology achieves up to 2× accuracy and up to 2× correct operations per unit energy compared to conventional CIM architectures.**

I. Introduction

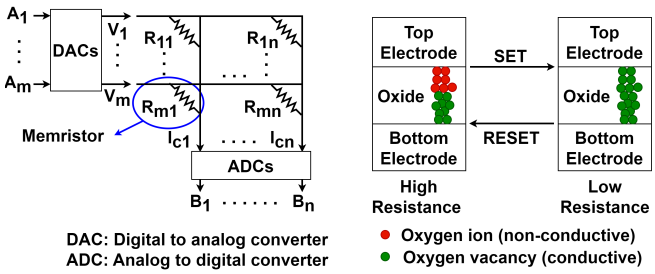
Neural networks form the backbone of modern artificial intelligence (AI) and are widely used for many cognitive tasks [1]. The implementation of neural networks using conventional von-Neumann architecture-based hardware, such as CPUs, GPUs, and AI-specific ASICs like TPUs [2–4], suffers from low energy efficiency due to the memory wall [5]. *Computation-in-memory* (CIM) offers a highly energy-efficient hardware alternative for neural network implementation [6, 7]. It performs computations within the memory to alleviate the memory wall problem, by using emerging non-volatile memory technologies such as memristors (also called resistive random access memories RRAMs) [8, 9]. Memristors are non-volatile, highly scalable, and compatible with CMOS technology. However, they suffer from various non-idealities that arise either at design-time or run-time, leading to computational errors. While design-time non-idealities can be fixed via circuit calibration, this approach cannot address run-time non-idealities. The most prominent run-time non-ideality affecting CIM-based neural networks is called read-disturb, where a large number of read operations lead to a significant resistance change in the memristor [10]. As neural network inference involves numerous read operations on memristors, read-disturb causes an undesired change in the weights stored as memristor resistances and results in degraded inference accuracy [11].

To reduce the impact of read-disturb, some works have recommended using low read voltages [10, 12, 13]. However, this leads to a reduced sensing margin and increased influence of process variation, resulting in erroneous output. Alternatively, CIM-aware training can be leveraged to address read-disturb. Such approaches fall into two categories: i) ex-situ where software models of non-idealities are incorporated into the training [14, 15], and ii) in-situ where training involves forward path execution directly on the CIM chip [16, 17]. The ex-situ approach only deals with design-time non-idealities, leaving run-time non-idealities like read-disturb unaddressed. The in-situ approach requires frequent on-chip training iterations, leading to high energy consumption and endurance issues. A few works provide only detection of the occurrence of read-disturb [18, 19], while [20, 21] recommend periodic re-programming of memristors for read-disturb mitigation which leads to excessive write energy. Some techniques periodically reverse the direction of the read current to compensate for the resistance change due to read-disturb [11, 22]. However, this strategy falls short due to the asymmetric behavior of read-disturb in opposite read directions, leading to incorrect compensation. Furthermore, it adds extra complexity to the hardware design, which can negatively affect inference accuracy in the presence of process variation. Hence, there is a strong need for an effective read-disturb mitigation technique to improve the accuracy of CIM-based neural networks.

In this paper, we present a methodology for dynamic detection and mitigation of read-disturb in CIM-based neural networks. It begins with an analysis to extract insights about the read-disturb phenomenon. We then develop a dynamic detection mechanism capable of identifying instances of read-disturb occurrence at run-time. Moreover, we propose an adaptive method based on the aforementioned analysis, that adjusts sensing conditions in CIM hardware upon detecting read-disturb to prolong the error-free operation. The key contributions of this paper are as follows:

- An analysis to derive insights about the read-disturb phenomenon in memristor-based neural networks.

This work is partially funded by the European Union, DAIS (Grant No. 101007273), CONVOLVE (Grant No. 101070374), NEUROKIT2E (Grant No. 101112268) and also supported by the TU Delft AI labs program.



(a) Vector-matrix multiplication. (b) Memristor technology.

Fig. 1. CIM-based vector-matrix multiplication [B]=[A]*[R].

- A mechanism for detecting the occurrence of read-disturb at run-time.
- An adaptive design to adjust the CIM sensing conditions to extend the correct functionality.

Our proposed methodology provides up to 2× accuracy and up to 2× correct operations per unit energy than conventional CIM architectures.

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

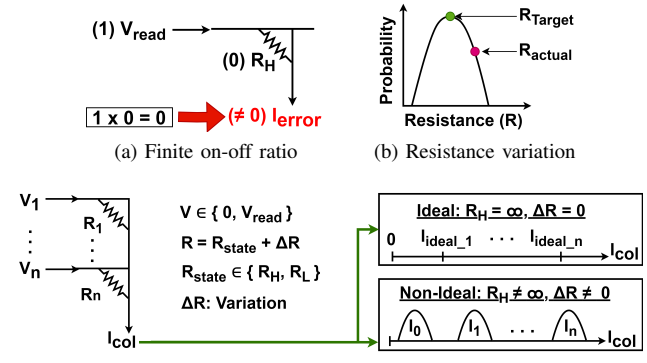
II. Background

A. Computation-in-memory Architecture for Neural Networks

Computation-in-memory (CIM) performs vector-matrix multiplication (VMM) within the memory as shown in Fig. 1a, to avoid frequent data transfers and save energy. Inputs are converted to voltages (V's) using digital-to-analog converters (DACs) and applied to weights stored as memristor resistances (R's). The resulting column currents (I_c's) denote a VMM operation. Analog-to-digital converters (ADCs) convert these currents to digital form for use by other system components.

B. Memristor Technology

Memristor, also called resistive random access memory (RRAM), consists of two metal electrodes separated by an oxide layer [6, 23] (Fig 1b). It stores data as 0 (high-resistance, R_H) and 1 (low-resistance, R_L). "SET" process changes its state from R_H to R_L by creating a conductive path with a set voltage, while "RESET" process shifts its state from R_L to R_H using a reset voltage. Data is retrieved via read operation by applying a read voltage and measuring resulting current to determine the resistance state of the memristor.



(a) Finite on-off ratio (b) Resistance variation

(c) Possible column current distributions at design-time.

Fig. 2. Design-time nonidealities and their impact.

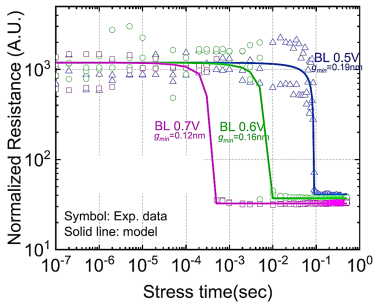


Fig. 3. Read-disturb phenomenon in memristors [11].

C. Memristor Non-idealities Affecting CIM-based Inference

1) Finite on-off ratio and resistance variation: Memristors represent a zero weight using non-zero resistance, known as finite on-off ratio [6, 24]. Programmed resistance of a memristor deviates from the target value due to stochastic physics and fabrication imperfections [6], called resistance variation. These non-idealities result in column current distributions due to deviation from ideal current as shown in Fig. 2.

2) Read-disturb: Although read operations on the memristor are performed with low voltages, they can still lead to a minuscule change in its resistance [10, 12]. Such minuscule changes accumulate into a substantial resistance change over numerous read operations, known as read-disturb, shown in Fig. 3 [11]. As neural network inference on CIM hardware is inherently a read-intensive task [11], read disturb changes weights stored as memristor resistances and degrades the neural network accuracy. In this paper, we improve the accuracy of CIM-based neural networks in the presence of read-disturb.

III. Proposed Methodology

A. Overview

Based on the experiments in [11], read disturb only impacts R_H memristors and decreases their resistance. Thus, the column currents increase over time and their distributions shift towards the right as shown in Fig. 4. Conventional design practice sets sensing reference at the midpoint between adjacent column current distributions, to optimize the sensing margin [25–27]. However, this leads to erroneous operation at

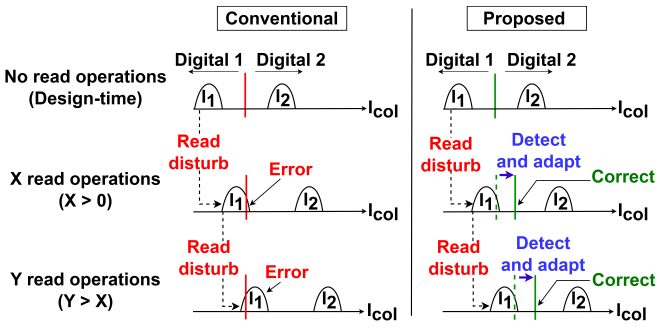


Fig. 4. Overview of proposed read-disturb detection and mitigation.

Algorithm 1: Identifying the most vulnerable columns.

input : Weights (W), inputs (I), fixed point weight format (F_W), fixed point input format (F_I), memristor bit-size (R), DAC resolution (D), crossbar size (X)
output: A matrix containing the location of the most vulnerable column in each crossbar (MVC_matrix)

$W_{fxp} \leftarrow$ fixed_point_conversion(W, F_W);
$W_{sliced} \leftarrow$ bit_slicing(W_{fxp}, R);
$W_{xbar} \leftarrow$ split_into_xbar_chunks(W_{sliced}, X);
$I_{fxp} \leftarrow$ fixed_point_conversion(I, F_I);
$I_{sliced} \leftarrow$ bit_slicing(I_{fxp}, D);
$I_{xbar} \leftarrow$ split_into_xbar_chunks(I_{sliced}, X);
RVS_matrix $\leftarrow$ track_nonzero_inputs_to_R_H(I_{xbar}, W_{xbar});
MVC_matrix $\leftarrow$ max(RVS_matrix, X);
return MVC_matrix

run-time due to right shift induced by read-disturb, as shown in Fig. 4. This challenge is also not effectively addressed by prior works as discussed in Section I. Our proposed methodology overcomes this problem by dynamically detecting the occurrences of read-disturb and then adapting the sensing references to restore correct operation as depicted in Fig. 4. It requires new hardware components such as read-disturb detection unit, ADC adaptation control, and adaptive ADC which are shown in Fig. 5 and discussed in the next subsection.

B. Implementation

1) Dynamic Read-disturb Detection: Dynamic read-disturb detection starts with software profiling in Algorithm 1, employing test data to identify the most vulnerable column (MVC) for read disturb. The weights and inputs are quantized, bit-sliced and divided into crossbar-compatible chunks. The number of read operations (N_{read}) on R_H memristors is obtained by tracking the instances where each digital 0 weight gets digital 1 input. The sum of N_{read} across memristors in a column gives its read-disturb vulnerability score (RVS). The column with the highest RVS in each XPE (Fig.5) becomes the MVC, monitored by its read-disturb detection unit (RDU) in Fig.6. RDU uses registers to store the MVC index and digital MVC weights, allowing monitoring of any crossbar column. It performs a multiply-accumulate operation (MAC) between the digital MVC weights and input register. A mismatch between MAC output and ADC output for MVC indicates read-disturb, activating the ADC adaptation control unit.

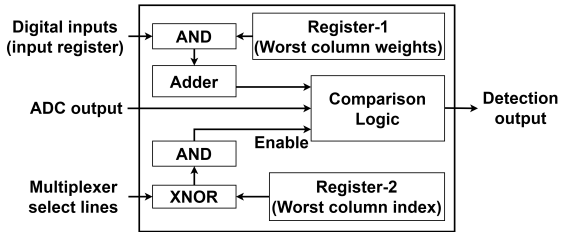


Fig. 6. Read-disturb detection unit.

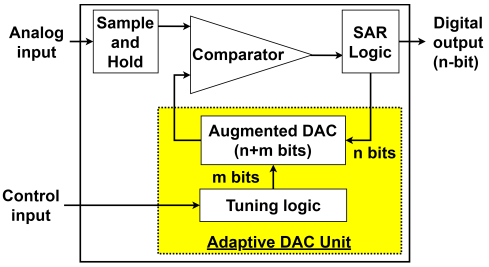


Fig. 7. Adaptive SAR ADC design.

2) Adaptive Sensing-based Mitigation: Successive approximation-register (SAR) ADC is widely used in CIM due to its low power consumption [28]. In n-bit SAR ADC, reference shifting is achieved by augmenting its internal DAC with m additional least significant bits (LSBs) and a tuning logic, as shown in Fig. 7. Upon activation, tuning logic increments m-bit LSB value by 1 and references shift by $V_{FS}/2^{(m+n)}$, where V_{FS} is the full-scale ADC voltage.

The ADC adaptation control unit in Fig. 8 tracks the number of executed reference shifts (C). When activated by RDU, it increments C by 1 if $C < 2^m$ and triggers the tuning logic in the adaptive ADC. It also sends a stall request to tile control logic, ensuring it waits for reference shifting completion. Conversely, $C = 2^m$ means no further shifting is possible and memristor reprogramming request is sent to tile control logic.

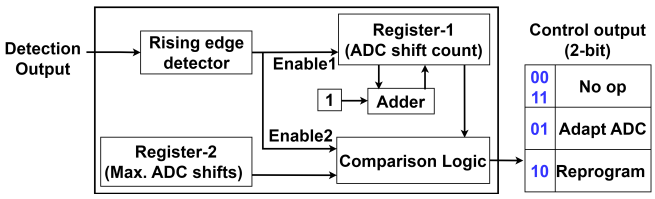


Fig. 8. ADC adaptation control unit.

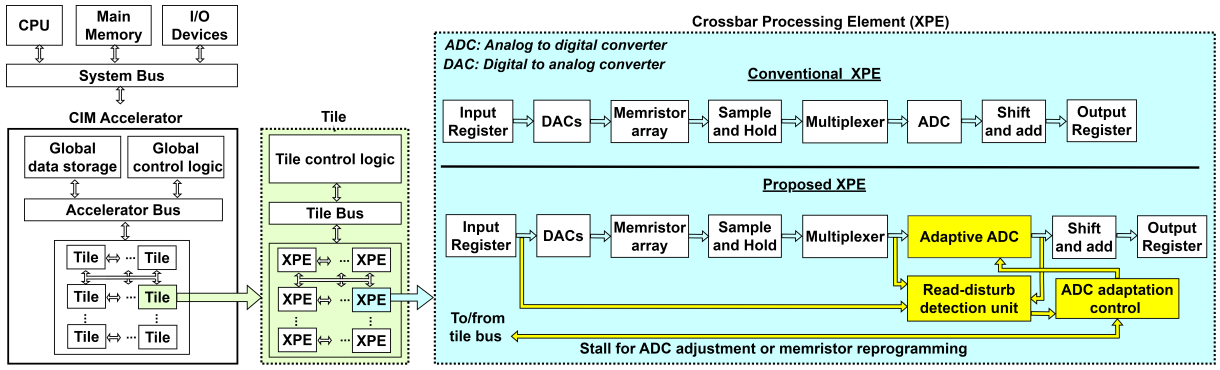
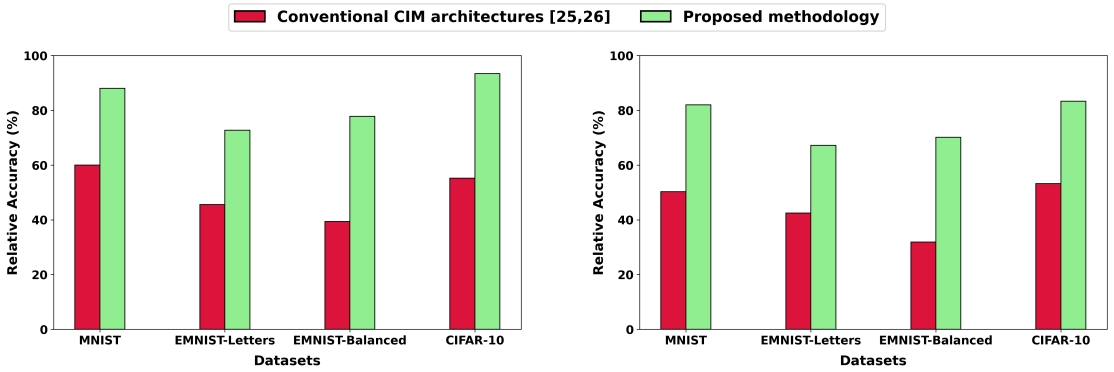


Fig. 5. CIM system architecture with conventional and proposed crossbar processing element (XPE), with new/modified components indicated in yellow.



(b) Read voltage 0.5V. (c) Read voltage 0.6V.

Fig. 9. Accuracy comparison between conventional CIM architectures [25, 26] and proposed methodology across various datasets and read voltages.

IV. Simulation Results

A. Simulation Setup

We use four datasets for evaluation: MNIST [29], EMNIST-Letters [30], EMNIST-balanced [30] and CIFAR-10 [31]. We modify the VGG [32] network for CIFAR-10 as: 32c3 $\rightarrow$ 32c3 $\rightarrow$ maxpool $\rightarrow$ 64c3 $\rightarrow$ 64c3 $\rightarrow$ maxpool $\rightarrow$ 128c3 $\rightarrow$ 128c3 $\rightarrow$ maxpool $\rightarrow$ flatten $\rightarrow$ 128 $\rightarrow$ 10. Here, nCm denotes a block of n filters of m kernel size with batch normalization and relu, while a single number indicates neurons in a fully connected layer with batch norm and relu (except the last layer). Lenet-5 [29] with batch normalization is used for all other datasets. After training these networks in PyTorch, we perform behavioral simulation of their inference on CIM hardware using our Python-based framework. This framework leverages the crossbar processing element (XPE) in Fig. 5 and adaptive ADC obtained by modifying the SAR ADC in [25, 26] as per Section III-B2 with four extra LSBs. Power and area for the adaptive ADC are obtained using [33]. We performed RTL synthesis in TSMC 40nm technology to derive power and area for read-disturb detection unit and ADC adaptation control unit. The power and area of other XPE components are obtained from [26]. We consider full-precision weights distributed across a group of 1-bit memristors, programmed using write-verify method in [34]. Read-disturb models are extracted from [11] which presents experimental investigations on real memristors. Our simulations consider both finite on-off ratio and resistance variation in addition to read-disturb.

B. Neural Network Accuracy

The accuracy comparison between proposed adaptive referencing and conventional CIM architectures [25, 26] is shown in Fig. 9. These results are presented in terms of relative accuracy, obtained by normalizing the accuracy of behavioral CIM hardware simulation with corresponding software baseline accuracy. This normalization effectively quantifies how faithfully computational correctness in software is maintained in CIM hardware. The proposed methodology achieves up to 2$\times$ accuracy compared to conventional CIM architectures (EMNIST-Balanced dataset with 0.5V read voltage), providing effective read-disturb mitigation. It also accommodates diverse dataset complexities (from MNIST to CIFAR-10) and adapts seamlessly to various network sizes (from Lenet-5 to VGG-like architecture). Furthermore, it delivers these benefits across two different read voltages, highlighting its robustness.

TABLE I
Comparison of crossbar processing element (XPE) metrics.

Metric	Conventional XPE [25, 26]	Proposed XPE [This work]
Energy consumption (pJ)	407.06	416.00
Net energy-efficiency (GOPS/W)	157.22	153.85
EMNIST-Balanced accuracy at 0.5V (%)	33.45	66.12
Correct operations per unit energy (GOP/J)	52.59	101.72
Area (μm^2)	3735.43	4617.82

C. Hardware Performance Evaluation

The hardware metrics per XPE for the proposed methodology and conventional CIM architectures [25, 26] are shown in Table I. Metrics like energy and net energy-efficiency expressed in giga operations per second per watt (GOPS/W) do not inherently account for the energy devoted to correct computations. Hence, we introduce a new metric called "correct operations per unit energy". It is defined as the ratio of correct operations to total energy consumption (unit: Giga operations per joule, GOP/J), where correct operations are given by the product of accuracy (as a fraction) and total operations. Our proposed methodology achieves up to 2$\times$ correct operations per unit energy at the expense of 2.2% energy overhead and 23.6% area overhead. This additional cost can be attributed to the increased resolution of the DAC within the SAR ADC, which is necessary to accommodate reference shifting. Thus, it is clear that its advantages outweigh the overheads.

V. Conclusion

This paper presented a read-disturb mitigation methodology to improve the accuracy of CIM-based neural networks, through run-time monitoring and adaptation of sensing references. It provides up to 2$\times$ accuracy than conventional CIM architectures. Thus, we showed that a shrewd design can facilitate correct operation despite memristor non-idealities.

REFERENCES

[1] M. Tan *et al.*, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019.

[2] E. Rotem *et al.*, "Intel Alder Lake CPU Architectures," *IEEE Micro*, 2022.

[3] J. Choquette *et al.*, "NVIDIA A100 Tensor Core GPU: Performance and Innovation," *IEEE Micro*, 2021.

[4] N. Jouppi *et al.*, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *International Symposium on Computer Architecture*, 2023.

[5] N. Verma *et al.*, "In-Memory Computing: Advances and Prospects," *Solid-State Circuits Magazine*, 2019.

[6] S. Yu *et al.*, "RRAM for Compute-in-Memory: From Inference to Training," *Transactions on Circuits and Systems I: Regular Papers*, 2021.

[7] A. Singh *et al.*, "A 115.1 TOPS/W, 12.1 TOPS/mm^2 Computation-in-Memory using Ring-Oscillator based ADC for Edge AI," in *International Conference on Artificial Intelligence Circuits and Systems*, 2023.

[8] X. Yang *et al.*, "Research Progress on Memristor: From Synapses to Computing Systems," *Transactions on Circuits and Systems I: Regular Papers*, 2022.

[9] R. Bishnoi *et al.*, "Energy-efficient Computation-In-Memory Architecture using Emerging Technologies," in *International Conference on Microelectronics*, 2023.

[10] W. Shim *et al.*, "Impact of Read Disturb on Multilevel RRAM based Inference Engine: Experiments and Model Prediction," in *International Reliability Physics Symposium*, 2020.

[11] W. Shim *et al.*, "Investigation of Read Disturb and Bipolar Read Scheme on Multilevel RRAM-Based Deep Learning Inference Engine," *Transactions on Electron Devices*, 2020.

[12] J. Reuben and S. Pechmann, "Accelerated Addition in Resistive RAM Array Using Parallel-Friendly Majority Gates," *Transactions on Very Large Scale Integration Systems*, 2021.

[13] Y. Lin *et al.*, "High-speed and High-efficiency Diverse Error Margin Write-Verify Scheme for an RRAM-based Neuromorphic Hardware Accelerator," *Transactions on Circuits and Systems II: Express Briefs*, 2022.

[14] D. Joksas *et al.*, "Nonideality-Aware Training for Accurate and Robust Low-Power Memristive Neural Networks," *Advanced Science*, 2022.

[15] G. Charan *et al.*, "Accurate Inference With Inaccurate RRAM Devices: A Joint Algorithm-Design Solution," *Journal on Exploratory Solid-State Computational Devices and Circuits*, 2020.

[16] P. Yao *et al.*, "Fully hardware-implemented memristor convolutional neural network," *Nature*, 2020.

[17] W. Ye *et al.*, "Aging Aware Retraining for Memristor-based Neuromorphic Computing," in *International Symposium on Circuits and Systems*, 2022.

[18] J.-H. Yoon *et al.*, "A 40nm 64Kb 56.67TOPS/W Read-Disturb-Tolerant Compute-in-Memory / Digital RRAM Macro with Active-Feedback-Based Read and In-Situ Write Verification," in *International Solid- State Circuits Conference*, 2021.

[19] M. A. Yaldagard *et al.*, "Read-disturb Detection Methodology for RRAM-based Computation-in-Memory Architecture," in *International Conference on Artificial Intelligence Circuits and Systems*, 2023.

[20] W. Li *et al.*, "A 40nm RRAM Compute-in-Memory Macro Featuring On-Chip Write-Verify and Offset-Cancelling ADC References," in *European Solid State Circuits Conference*, 2021.

[21] W. Li *et al.*, "A 40-nm MLC-RRAM Compute-in-Memory Macro With Sparsity Control, On-Chip Write-Verify, and Temperature-Independent ADC References," *Journal of Solid-State Circuits*, 2022.

[22] B. Yan *et al.*, "A closed-loop design to enhance weight stability of memristor based neural network chips," in *International Conference on Computer-Aided Design*, 2017.

[23] A. Gebregiorgis *et al.*, "Dealing with Non-Idealities in Memristor Based Computation-In-Memory Designs," in *International Conference on Very Large Scale Integration*, 2022.

[24] S. Diware *et al.*, "Accurate and Energy-Efficient Bit-Slicing for RRAM-Based Neural Networks," *Transactions on Emerging Topics in Computational Intelligence*, 2023.

[25] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *International Symposium on Computer Architecture*, 2016.

[26] A. Ankit *et al.*, "PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[27] Y. Park *et al.*, "RM-NTT: An RRAM-Based Compute-in-Memory Number Theoretic Transform Accelerator," *Journal on Exploratory Solid-State Computational Devices and Circuits*, 2022.

[28] W. Wan *et al.*, "A compute-in-memory chip based on resistive random-access memory," *Nature*, 2022.

[29] Y. Lecun *et al.*, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, 1998.

[30] G. Cohen *et al.*, "EMNIST: Extending MNIST to handwritten letters," in *International Joint Conference on Neural Networks*, 2017.

[31] The CIFAR-10 dataset. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[32] K. Simonyan *et al.*, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2015.

[33] M. Saberi *et al.*, "Analysis of Power Consumption and Linearity in Capacitive Digital-to-Analog Converters Used in Successive Approximation ADCs," *Transactions on Circuits and Systems I: Regular Papers*, 2011.

[34] Y. Luo *et al.*, "Array-Level Programming of 3-Bit per Cell Resistive Memory and Its Application for Deep Neural Network Inference," *Transactions on Electron Devices*, 2020.