2025 IEEE International Symposium on Circuits and Systems (ISCAS)

Efficient Reprogramming of Memristive Crossbars for DNNs: Weight Sorting and Bit Stucking

Matheus Farias
Harvard University
matheusfarias@g.harvard.edu

H. T. Kung
Harvard University
kung@harvard.edu

***Abstract*—We introduce a novel approach to reduce the number of times required for reprogramming memristors on bit-sliced compute-in-memory crossbars for deep neural networks (DNNs). Our idea addresses the limited non-volatile memory endurance, which restricts the number of times they can be reprogrammed.**

To reduce reprogramming demands, we employ two techniques: (1) we organize weights into sorted sections to schedule reprogramming of similar crossbars, maximizing memristor state reuse, and (2) we reprogram only a fraction of randomly selected memristors in low-order columns, leveraging their bit-level distribution and recognizing their relatively small impact on model accuracy.

We evaluate our approach for state-of-the-art models on the ImageNet-1K dataset. We demonstrate a substantial reduction in crossbar reprogramming count by 3.7x for ResNet-50 and 21x for ViT-Base, while maintaining model accuracy within a 1% margin.

***Index Terms*—memristive crossbars, compute-in-memory, efficient reprogramming.**

I. Introduction

Resistive compute-in-memory (CIM) crossbars offer a promising computing architecture for deep neural networks (DNNs), using analog accelerators to achieve fast, power-efficient matrix multiplication. Central to this architecture are memristors, which enable simultaneous data storage and processing, overcoming the data movement limitations of traditional von Neumann systems [1]–[7].

DNN weights are encoded into memristor conductances by applying sequential voltage pulses, which significantly increases programming time. In conventional 1 transistor–1 memristor (1T1R) crossbar architectures, each memristor is programmed individually, with the transistor acting as a switch to isolate the target cell for precise conductance adjustment [8]–[14].

Memristors, based on non-volatile memory technologies [4], [5], [15], offer high retention but have limited endurance, degrading after a certain number of reprogramming cycles. In CIM architectures, where read/write operations are frequent, endurance is especially critical; for instance, a 1024x1024 resistive RAM crossbar used for 32-bit multiplication may fail within minutes [16]. This issue is intensified when implementing large DNNs, as they require partitioned sections to fit onto relatively small, fixed-size crossbars. Consequently, each crossbar requires multiple reprogramming cycles to accommodate each partition, further straining endurance. We propose programming sorted weight sections to reduce reprogramming frequency, thereby extending the lifespan of memristor-based architectures without sacrificing efficiency.

Using sorted sections minimizes memristor state mismatch during reprogramming by grouping crossbars that store similar weights. To the best of our knowledge, this is the first implementation of Sorted Weight Sectioning (SWS) [17] to minimize crossbar reprogramming. Additionally, SWS enhances parallelization by distributing programming across multiple crossbars based on weight magnitude, balancing the workload across threads and further minimizing reprogramming. Previously, SWS was explored primarily for improving energy efficiency, an orthogonal benefit of the approach.

Further, we propose a reprogramming method that leverages the distribution of memristor states and the statistical properties of bitline-represented numbers. As bitwidth increases, the probability of the least significant bit being one approaches 50%, indicating inherent randomness in lower-order bits. From a CIM perspective, this reveals a lack of correlation between the bell-shaped distribution of DNN [18]–[21] and the uniform distribution of active memristors in low-order columns, which are farther from the input and contribute minimally to the overall weight magnitude. Recognizing their limited impact, we selectively reprogram only a small fraction of memristors in the lowest-order column, reducing the overall reprogramming workload without sacrificing performance.

See Figure 1 for a summary of the approach. The main contributions of this paper are:

- The use of sorted weight sectioning to (1) minimize reprogramming and (2) efficient scheduling of multiple crossbar programming without penalizing model accuracy
- A bit stucking procedure based on the uniform distribution of memristors in low-order columns to further reduce reprogramming with minimal DNN accuracy loss

II. Background and Related Work

Past works addressed crossbar programming efficiency in terms of time and accuracy. Feedback strategies [14], [22] use multiple read/write operations to approximate the output of multi-level memristors. Variable amplitude pulses [8], [11], [23] implement finer transitions between adjacent conductance levels. Emmanuelle et al. [13] combine feedback and variable pulses, representing the state-of-the-art in sequential programming. Recent papers addressed parallel memristor programming with sequential fine-tuning [14] and neural networks assisting memristor programming [24].

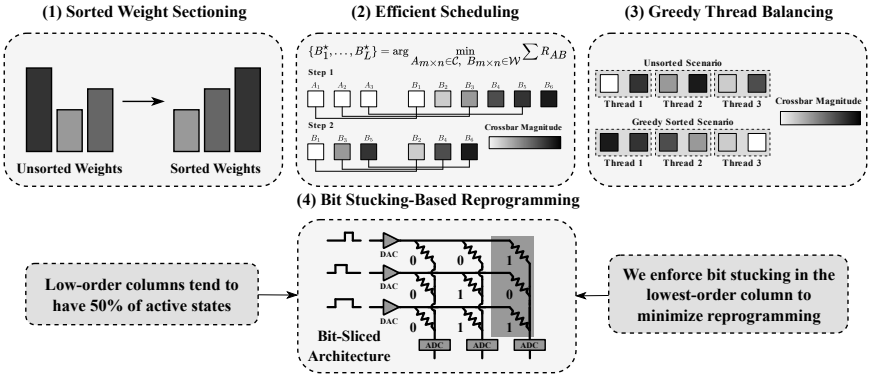

Fig. 1: Summary of the approach. (1) We apply sorted weight sectioning, (2) efficiently schedule crossbars with stride 1 to minimize transitional states, (3) assign similar crossbars to each thread in a greedy approach to minimize imbalance, and (4) since low-order columns tend to have 50% of active states, we enforce bit stucking in the lowest-order column to minimize reprogramming without severe DNN accuracy degradation.

Our work targets bit-sliced crossbars, where each row corresponds to a single weight value and each column represents a power-of-two multiplier. For example, in a 128x128 crossbar with 16 power-of-two multipliers, we can represent $128/16 = 8$ weights per row. We label crossbars as 128x16 instead of 128x128 with 16 multipliers. Additionally, we do not implement feedback loops since bit-slicing already ensures accurate weight representation without the overhead of time-consuming read/write operations [5]. Furthermore, we use constant amplitude phases to avoid the higher power consumption of variable amplitude phases.

III. Optimizing Reprogramming with Sorted Weight Sectioning

In bit-sliced crossbars, reprogramming is limited to memristors that change states. To minimize transitions, Sorted Weight Sectioning (SWS) sorts weights by magnitude and groups them into sections, reducing the frequency of state changes. Our approach, while utilizing SWS, is fundamentally distinct, with orthogonal benefits. Sorting is done offline, but inference requires buffers for batching and index matching, introducing hardware overhead analyzed in [17]. Our focus is on reducing reprogramming and improving energy efficiency through a separate optimization strategy.

A. Reprogramming Crossbars

Consider we have two crossbars $A, B \in \mathbb{R}_{m\times n}$, with memristors $a_{ij}, b_{ij} \in \{0, 1\}$ representing, respectively, inactive and active states, for $i \in \mathbb{R}_m$ and $j \in \mathbb{R}_n$. The reprogramming cost to convert from A to B is

$$R_{AB} = \sum_{i,j} |a_{ij} - b_{ij}|. \tag{1}$$

If $|a_{ij} - b_{ij}| = 0$, the memristor a_{ij} does not face a transitional state. On the other hand, if $|a_{ij} - b_{ij}| = 1$, we switch states. Mathematically, we want to find $B^\star$ such that

$$B^\star = \arg \min_{B_{m\times n}\in\mathcal{W}} R_{AB}, \tag{2}$$

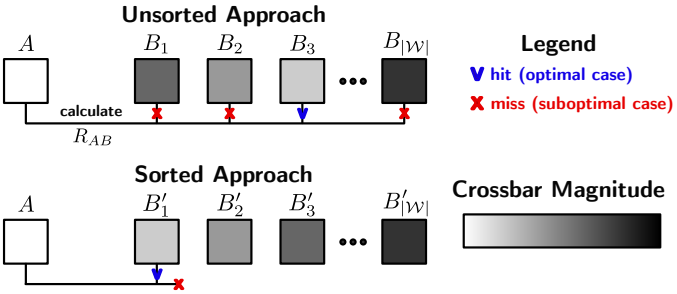

Fig. 2: The unsorted approach calculates reprogramming costs for all sections, while the sorted approach picks the next crossbar to reduce state mismatches.

where $\mathcal{W}$ is the set of all weight matrix sections of size $m \times n$.

Without SWS, the optimal scenario involves iterating through sections to find the lowest cost. With SWS, we simply program the next section in the sorted list (see Figure 2).

B. Extending to Multiple Crossbars

Now suppose we have a set $\mathcal{C}$ of L programmable crossbars. Our mathematical optimization model becomes

$$\{B_1^\star, \ldots, B_L^\star\} = \arg \min_{A_{m\times n}\in\mathcal{C},\ B_{m\times n}\in\mathcal{W}} \sum R_{AB}. \tag{3}$$

That is, we want to find a collection of matrix sections $\{B_1^\star, \ldots, B_L^\star\}$ that minimizes reprogramming of L crossbars.

We propose the stride L and the stride 1 scheduling.

Stride L Scheduling. We select the L first crossbars in the sorted list, At each reprogramming time, we program the crossbar $i + L$ to the crossbar i.

Stride 1 Scheduling. We select L evenly spaced crossbars in the sorted list, At each reprogramming time, we program the crossbar $i + 1$ to the crossbar i.

The stride L efficiency depends on the skip length L and the similarity of sections in the sorted list. Although stride 1 initially incurs higher costs by programming the

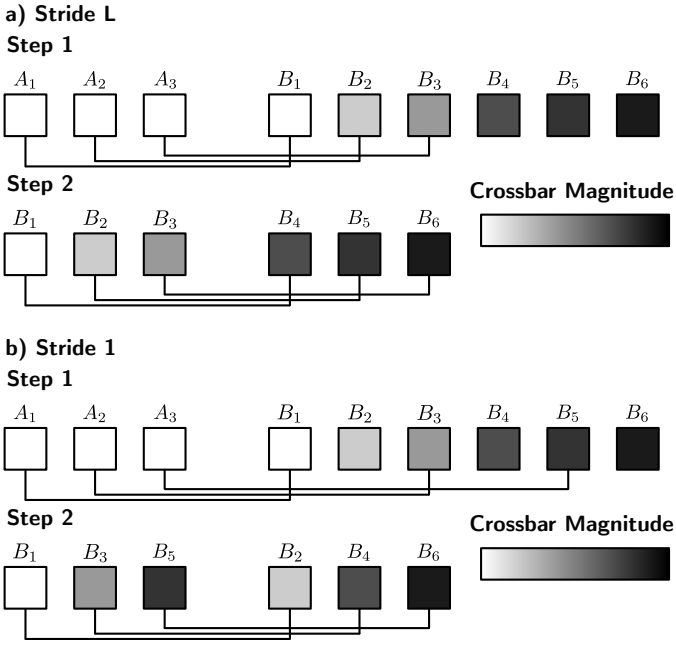

Fig. 3: Different scheduling methods for programming multiple crossbars a) stride L and b) stride 1.

first L crossbars, it only skips one crossbar at each step, making it advantageous over multiple reprogramming tasks, as demonstrated in Figure 3. In CIM, read/write operations for multiplication occur over 150x more frequently than in conventional systems, significantly straining device endurance. For example, a 1024x1024 resistive RAM crossbar performing 32-bit multiplication can fail in about five minutes [16]. Additionally, the limited physical space in crossbars prevents storing entire DNN weight matrices, requiring frequent reprogramming to load different sections. Under these conditions, stride 1 reduces reprogramming effectively, as discussed in Section V.

C. Multiple Crossbar Programming Thread Imbalance

If we program L crossbars in parallel, we expect an L-fold speedup. However, the actual speedup depends on workload balancing across threads. If a thread handles both small and large reprogramming costs, the overall programming time may be bottlenecked by the highest reprogramming cost.

To optimize parallel programming of multiple crossbars, we propose a greedy approach based on SWS. We group crossbars of similar reprogramming costs on each thread (see Figure 4).

IV. Bit Stucking-Based Reprogramming

Now we leverage bit-level weight distribution to further reduce reprogramming with minimal DNN accuracy loss.

Consider a random sample of numbers in bitline representation. The probability that a number has a digit one on its least significant bit (LSB) tends to 50% as we increase the bitwidth: the distribution of values in the LSB is approximately uniform.

Bit-sliced crossbars map weights in bitlines along each row, with the LSB stored in the lowest-order column memristor. Since each column is a power-of-two multiplier, its distance

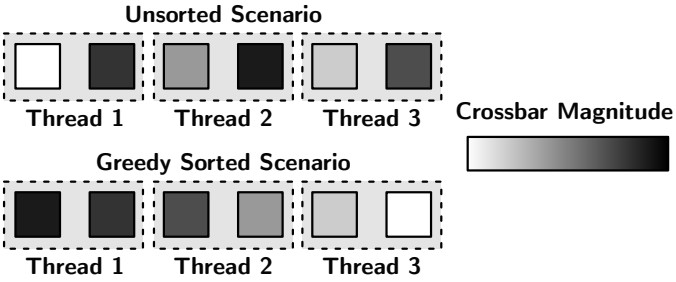

Fig. 4: We group crossbars of similar cost on each thread to boost speedup when parallel programming multiple crossbars.

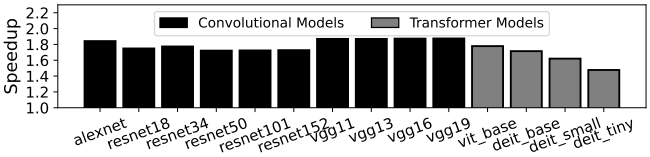

Fig. 5: Speedup of SWS for a single 128x16 crossbar.

from the input determines its contribution to the weight magnitude – the lowest-order column is the smallest multiplier.

Unstructured pruning is a compressing technique that sets weights below a certain threshold to zero [25]. For CIM crossbars, a similar concept is applied stucking column outputs. In this case, weights are not set to zero; instead, we introduce errors. Notably, altering the lowest-order column impacts all crossbar weights, thus, harsher than unstructured pruning.

Depending on the number of columns, pruning the lowest-order column might have a severe impact on the final accuracy. Noteworthy, low-order columns tend to have more transitional memristors due to their uniform distribution. We provide a DNN accuracy analysis when neglecting some transitional memristors in the last column to understand the reprogramming tradeoff.

V. Experiments

We assess performance using speedup (ratio of memristors that needed to switch states) and model accuracy, benchmarking against the state-of-the-art unsorted scenarios from CASCADE [26] and ISAAC [3]. Crossbar computations were simulated in PyTorch on ImageNet-1K [27] on all model layers (ResNets and VGGs from PyTorch, and ViTs and DeITs from timm), trained in 32-bit floating point precision. Simulations utilized 128x10 crossbars unless specified otherwise.

A. Sorted Weight Sectioning on a Single Crossbar

The DNN bell-shaped distribution [18]–[21] results in many crossbars with low-magnitude weights, which require fewer transitional states. The gradual transition from small to large-magnitude crossbars is essential to understanding how SWS optimizes reprogramming. In Figure 5, DeIT-Tiny, with its sharp weight distribution, achieved the lowest speedup (1.47x), while VGGs, with smoother distributions, saw higher improvements, reaching 1.87x on VGG16. Notably, SWS enhanced programming speed for all models.

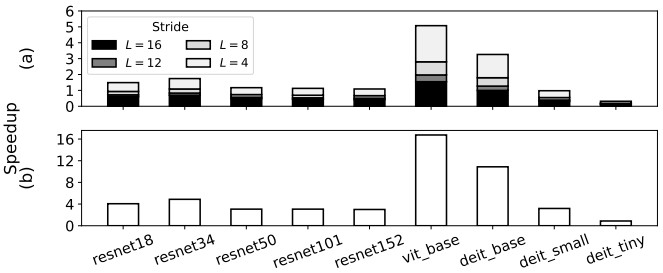

Fig. 6: Speedup of parallel programming multiple crossbars with (a) stride L and (b) stride 1 scheduling methods.

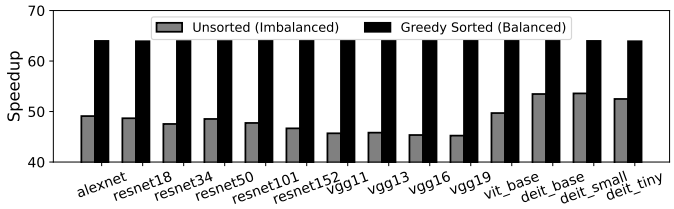

Fig. 7: Speedup of greedy approach with 64 crossbar threads.

B. Sorted Weight Sectioning on Multiple Crossbars

We compare average speedup per crossbar using 16 reprogrammable crossbars and observe that increasing stride reduces speedup (Figure 6). Larger strides skip multiple (L) crossbars per reprogramming, slowing progress. Stride 1 performs best; ViT-Base achieves 3x speedup over $L = 4$, indicating that programming farther crossbars becomes negligible over time.

Now, we analyze the parallel programming of multiple crossbars. The unsorted result observed in Figure 7 is bottlenecked by the slowest crossbars in each thread. We note that VGG models suffered more with thread imbalance due to the disparity between crossbars on each thread. With the greedy method, we obtained a result very close to the ideal of 64x speedup.

C. Bit Stucking-Based Reprogramming Analysis

The bit stucking-based reprogramming experiments, shown in Figure 8, compare $p = 1$ (full reprogramming of necessary memristors in the lowest-order column) with $p = 0.5$ (only half are reprogrammed). Speedups ranged from 19% for AlexNet to 27% for DeIT-Base, with minimal accuracy loss (less than 1%). A further sweep of p values for ViT-Base and ResNet-50 in Figure 9 demonstrates that reducing p down to 0—essentially stucking the last column—maintains accuracy within 1%. Overall, tuning p between 0 and 1 effectively balances speedup and accuracy.

Finally, we fix $p = 0.5$ and sweep the number of columns (see Figure 10). Sweeping columns provided almost constant speedup, while the accuracy reaches a plateau in 10 columns (78.00% and 80.31% in ViT-Base and ResNet-50, respectively).

The accuracy drop before stabilization occurs because reducing crossbar columns lowers weight bitwidths, and stucking low-order columns significantly affects the bitline

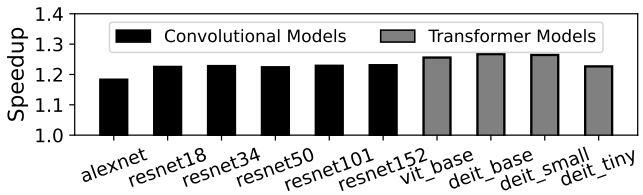

Fig. 8: Speedup with $p = 0.5$ over $p = 1$.

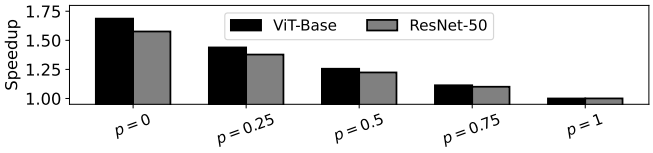

Fig. 9: Speedup sweeping p for ViT-Base and ResNet-50.

representation for lower bitwidths. Nonetheless, bit stucking-based reprogramming using SWS at stride 1 and $p = 0.5$ achieved substantial speedups—3.7x for ResNet-50 and 21x for ViT-Base—while maintaining less than 1% accuracy drop.

VI. Conclusion

We showed that sorting pretrained DNN weights for bit-sliced CIM crossbars and leveraging memristor distribution in low-order columns significantly save reprogramming time by minimizing transitional resistance states. The method's effectiveness depends on how parameters are distributed in crossbars: bit-level sparsity, models with similar matrices, and large-sized crossbars provide better results.

We validate these results on ImageNet-1K dataset. We achieved substantial speedups in crossbar programming—3.7x for ResNet-50 and 21x for ViT-Base—while incurring less than 1% accuracy drop. This work suggests a new research direction on efficient crossbar reprogramming based on weight placement on crossbars.

VII. Acknowledgements

This research was supported in part by the Air Force Research Laboratory under award number FA8750-22-1-0500, and in part by Meta Platforms Technologies under award number A51540.

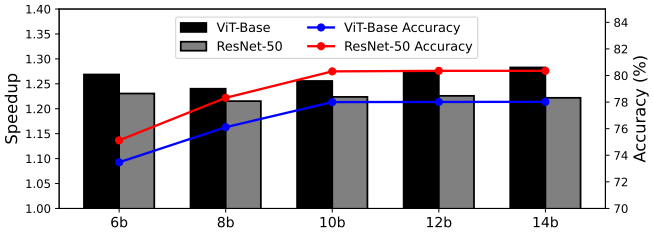

Fig. 10: Speedup and accuracy of bit stucking-based reprogramming sweeping number of crossbar columns for ViT-Base and ResNet-50 with probability $p = 1$ over $p = 0.5$.

REFERENCES

[1] O. Mutlu, S. Ghose, J. Gómez-Luna, R. Ausavarungnirun, M. Sadrosadati, and G. F. Oliveira, *A Modern Primer on Processing in Memory*, pp. 171–243. Singapore: Springer Nature Singapore, 2023.

[2] I. Chakraborty, M. Ali, A. Ankit, S. Jain, S. Roy, S. Sridharan, A. Agrawal, A. Raghunathan, and K. Roy, "Resistive Crossbars as Approximate Hardware Building Blocks for Machine Learning: Opportunities and Challenges," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2276–2310, 2020.

[3] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 14–26, 2016.

[4] S. Jung, H. Lee, S. Myung, H. Kim, S. K. Yoon, S.-W. Kwon, Y. Ju, M. Kim, W. Yi, S. Han, B. Kwon, B. Seo, K. Lee, G.-H. Koh, K. Lee, Y. Song, C. Choi, D. Ham, and S. J. Kim, "A crossbar array of magnetoresistive memory devices for in-memory computing," *Nature*, vol. 601, no. 7892, pp. 211–216, 2022.

[5] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[6] Q. Huo, Y. Yang, Y. Wang, D. Lei, X. Fu, Q. Ren, X. Xu, Q. Luo, G. Xing, C. Chen, X. Si, H. Wu, Y. Yuan, Q. Li, X. Li, X. Wang, M.-F. Chang, F. Zhang, and M. Liu, "A computing-in-memory macro based on three-dimensional resistive random-access memory," *Nature Electronics*, vol. 5, no. 7, pp. 469–477, 2022.

[7] X. Zou, S. Xu, X. Chen, L. Yan, and Y. Han, "Breaking the von Neumann bottleneck: architecture-level processing-in-memory technology," *Science China Information Sciences*, vol. 64, p. 160404, Apr 2021.

[8] D. Kuzum, R. G. D. Jeyasingh, B. Lee, and H.-S. P. Wong, "Nanoelectronic Programmable Synapses Based on Phase Change Materials for Brain-Inspired Computing," *Nano Letters*, vol. 12, p. 2179–2186, May 2012.

[9] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High-precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, p. 075201, Feb. 2012.

[10] S. B. Eryilmaz, D. Kuzum, R. Jeyasingh, S. Kim, M. BrightSky, C. Lam, and H.-S. P. Wong, "Brain-like associative learning using a nanoscale non-volatile phase change synaptic device array," *Frontiers in Neuroscience*, vol. 8, 2014.

[11] L. Gao, P.-Y. Chen, and S. Yu, "Programming Protocol Optimization for Analog Weight Tuning in Resistive Memories," *IEEE Electron Device Letters*, vol. 36, p. 1157–1159, Nov. 2015.

[12] L. Gao, I.-T. Wang, P.-Y. Chen, S. Vrudhula, J.-S. Seo, Y. Cao, T.-H. Hou, and S. Yu, "Fully parallel write/read in resistive synaptic array for accelerating on-chip learning," *Nanotechnology*, vol. 26, p. 455204, Nov. 2015.

[13] E. J. Merced-Grafals, N. Dávila, N. Ge, R. S. Williams, and J. P. Strachan, "Repeatable, accurate, and high speed multi-level programming of memristor 1t1r arrays for power efficient analog computing applications," *Nanotechnology*, vol. 27, p. 365202, Sept. 2016.

[14] G. L. Zhang, B. Li, X. Huang, C. Shen, S. Zhang, F. Burcea, H. Graeb, T.-Y. Ho, H. Li, and U. Schlichtmann, "An Efficient Programming Framework for Memristor-based Neuromorphic Computing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (Grenoble, France), p. 1068–1073, IEEE, Feb. 2021.

[15] C. Kugeler, C. Nauenheim, M. Meier, A. Rudiger, and R. Waser, "Fast resistance switching of TiO2 and MSQ thin films for non-volatile memory applications (RRAM)," in *2008 9th Annual Non-Volatile Memory Technology Symposium (NVMTS)*, (Pacific Grove, CA, USA), pp. 1–6, IEEE, 2008.

[16] S. Resch, H. Cilasun, Z. Chowdhury, M. Zabihi, Z. Zhao, J.-P. Wang, S. Sapatnekar, and U. R. Karpuzcu, "On Endurance of Processing in (Nonvolatile) Memory," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, (New York, NY, USA), Association for Computing Machinery, 2023.

[17] M. Farias and H. T. Kung, "Sorted Weight Sectioning for Energy-Efficient Unstructured Sparse DNNs on Compute-in-Memory Crossbars," *arXiv:2410.11298 [cs.AR]*, 2024.

[18] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *International Conference in Learning Representations (ICLR)*, 2016.

[19] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, and J. Hassoun, "Post-Training Piecewise Linear Quantization for Deep Neural Networks," in *The European Conference on Computer Vision (ECCV)*, 2020.

[20] M. Horton, Y. Jin, A. Farhadi, and M. Rastegari, "Layer-Wise Data-Free CNN Compression," in *International Conference on Pattern Recognition (ICPR)*, 2022.

[21] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. Janapa Reddi, A. Rush, D. Brooks, and G.-Y. Wei, "Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[22] J. Yu, P.-P. Manea, S. Ameli, M. Hizzani, A. Eldebiky, and J. P. Strachan, "Analog Feedback-Controlled Memristor Programming Circuit for Analog Content Addressable Memory," in *2023 IEEE International Conference on Metrology for eXtended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE)*, pp. 983–988, 2023.

[23] F. Corinto and M. Forti, "Memristor Circuits: Pulse Programming via Invariant Manifolds," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 4, pp. 1327–1339, 2018.

[24] Z. Yu, M.-J. Yang, J. Finkbeiner, S. Siegel, J. P. Strachan, and E. Neftci, "The Ouroboros of Memristors: Neural Networks Facilitating Memristor Programming," in *2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, pp. 398–402, 2024.

[25] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and Quantization for Deep Neural Network Acceleration: A Survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[26] T. Chou, W. Tang, J. Botimer, and Z. Zhang, "CASCADE: Connecting RRAMs to Extend Analog Dataflow In An End-To-End In-Memory Processing Paradigm," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, p. 114–125, 2019.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.