IMSSA: Deploying modern state-space models on memristive in-memory compute hardware

1st Sebastian Siegel
Peter-Gruenberg-Institute (PGI-14)
Forschungszentrum Juelich GmbH
Juelich, Germany
s.siegel@fZ-juelich.de

2nd Ming-Jay Yang
Peter-Gruenberg-Institute (PGI-14)
Forschungszentrum Juelich GmbH
Juelich, Germany
m.yang@fz-juelich.de

3rd John-Paul Strachan
Peter-Gruenberg-Institute (PGI-14)
Forschungszentrum Juelich GmbH
Juelich, Germany
j.strachan@fz-juelich.de

***Abstract*—Processing long temporal sequences is a key challenge in deep learning. In recent years, Transformers have become state-of-the-art for this task, but suffer from excessive memory requirements due to the need to explicitly store the sequences. To address this issue, structured state-space sequential (S4) models recently emerged, offering a fixed memory state while still enabling the processing of very long sequence contexts. The recurrent linear update of the state in these models makes them highly efficient on modern graphics processing units (GPU) by unrolling the recurrence into a convolution. However, this approach demands significant memory and massively parallel computation, which is only available on the latest GPUs.**

In this work, we aim to bring the power of S4 models to edge hardware by significantly reducing the size and computational demand of an S4D model through quantization-aware training, even achieving ternary weights for a simple real-world task. To this end, we extend conventional quantization-aware training to tailor it for analog in-memory compute hardware. We then demonstrate the deployment of recurrent S4D kernels on memrisitve crossbar arrays, enabling their computation in an in-memory compute fashion. To our knowledge, this is the first implementation of S4 kernels on in-memory compute hardware.

***Index Terms*—state-space models, in-memory computing, memristive crossbar arrays**

I. Introduction

Processing long temporal sequences presents a significant challenge for many deep learning algorithms [1], [2]. Transformers [3] address this by explicitly storing projections of a long history of input tokens [4]. However, this dynamic allocation of memory, which scales quadratically with the sequence length [5], results in substantial memory requirements. Consequently, these algorithms become impractical for resource-constrained applications, such as natural language processing in off-grid environments or remote sensor data processing.

A promising solution to this issue is the emergence of state-space sequential models, such as S4(D) [11] and MAMBA [12]. They overcome the training challenges associated with classical recurrent neural networks by employing a linear state transition, which can be unrolled into a convolutional kernel. This allows for efficient training and execution on modern GPUs, which have the memory capacity to store the entire convolutional kernel. However this approach is impractical on edge computing hardware, where memory is limited. An alternative is to perform the state update in a recurrent manner, which reduces memory requirements but necessitates an efficient method for computing vector-matrix multiplications (VMM).

An efficient choice for such operations are memristive crossbar arrays (MCBA) [14]–[20] which offer analog in-memory computing of VMMs. MCBAs use emerging non-volatile resistive switching devices and have been used for various VMM accelerators, because they allow computing a VMM in a single operation [28]–[31]. However, to our knowledge, their application to modern state-space models has not been investigated so far, which is the aim of this work.

In this manuscript, we first illustrate the hardware-aware training process required for successfully deploying the recurrent kernels of an S4D model on an MCBA. To address this, we incorporate the limited dynamic range of memristive devices into a quantization-aware training approach. We also introduce the In-Memory State-Space model Accelerator architecture (IMSSA), which maps an entire S4D kernel onto a single MCBA, and we demonstrate its performance on a simple real-world task. Finally, we highlight the critical role of quantization in enabling the effective deployment of state-space kernels onto noisy computational substrates.

II. Methods

A. The S4(D) model

At the core of the S4(D) model are the so-called HiPPO kernels, originally proposed by Gu et al. [6]. These kernels use a vector $\mathbf{B} \in \mathbb{R}^N$ to project a one-dimensional input signal u(t) into the higher dimension of the state x(t) $\in \mathbb{R}^N$. The state is recurrently updated via the projection matrix $\mathbf{A} \in \mathbb{R}^{NxN}$ and the output of each kernel is calculated by multiplying the state with a vector $\mathbf{C} \in \mathbb{R}^N$. While it is possible to include a skip connection directly from the input to the output, this is not implemented in the current work. The complete kernel can be formulated as

$$\begin{aligned} \frac{dx(t)}{dt} &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t). \end{aligned} \tag{1}$$

Identify applicable funding agency here. If none, delete this.

In time-discrete systems, for example for training on conventional GPUs, the kernel must be discretized in time. A commonly used method, which is also applied in this work, is the zero-order hold method. For a constant time-step Δ, which is a trainable parameter, this approach results in new time-discrete kernels

$$\begin{aligned} x_t &= \overline{\mathbf{A}} x_{t-1} + \overline{\mathbf{B}} u_t \\ y_t &= \overline{\mathbf{C}} x_t. \end{aligned} \tag{2}$$

An S4 layer consists of H kernels running in parallel, combined with a linear mixing layer. The mixing layer projects the outputs of the HiPPO kernels back to the input dimension H of the next layer. Most S4 models are composed of multiple such layers stacked in series, typically enclosed by linear encoder and decoder layers.

Since these models are trained on conventional GPUs, the recurrent vector-matrix multiplication imposes a computational bottleneck. In addition to unrolling the recurrence into a convolution, an important step is the diagonalization of the matrix $\mathbf{A}$. This results in a complex matrix, which in turn leads to a complex state x(t) and a complex matrix $\mathbf{C}$. Models utilizing this approach are referred to as S4D models.

B. Quantization-aware training

For deployment on low-resolution inference hardware, the parameters of the S4D model typically need to be stored and computation executed using quantized integer formats. However, the model is trained on high-precision floating point GPUs. While post-training quantization of the high-precision model is possible, it significantly reduces accuracy, even at moderate quantization levels. To achieve more aggressive quantization, a common technique is quantization-aware training (QAT). Abreu et al. [7] have demonstrated the benefits with this approach on a similar state-space model (S5).

A common approach to QAT in auto-differentiating machine learning frameworks is the use of the straight-through estimator. In this method, the forward pass of the model employs quantized parameters and activations, while the backward pass and gradient updates are performed with full precision. This approach has been successfully applied to numerous recent QAT tasks [7], [13]. As an extension of this method, we introduce the training for a specific constant dynamic range by choosing a constant f_{scale} in the quantization function

$$x_{i,quant} = round\left(x_i * \frac{n_{levels}}{f_{scale}}\right) * \frac{f_{scale}}{n_{levels}}. \tag{3}$$

C. Task description

The target application of the S4D model discussed here involves classification tasks at the edge, such as key word spotting. To evaluate its performance, we tested the model on a two-class subset of the Heidelberg Digits raw audio dataset. Specifically, the model is trained to distinguish between vocal utterances of the English words "zero" and "one". The audio files are normalized and downsampled by a factor of 64, resulting in 871 samples per input.

D. Memristive crossbar arrays

Memristive devices are an emerging class of non-volatile memory elements, functioning as two-terminal resistors with electronically adjustable conductance. Their conductance can be programmed gradually, enabling their use in constant time VMM through Ohm's Law and Kirchhoff's Law. To achieve this, memristive devices are typically arranged in matrix-like structures (see Figure 1) to perform the multiplication of an applied voltage vector with a stored conductance matrix in a single operation, in contrast to the quadratically scaling complexity of this operation on classical digital hardware. Memristive devices are often put in series with a transistor for more precise programming. Once programmed, these devices behave like passive resistors and can only represent real-valued positive conductances. However, S4D kernels are complex-valued and can have both positive and negative real and imaginary components. As a result, each element of the kernels must first be expanded similar to [32] into its real and imaginary parts for further processing,

$$\underline{m} * \underline{v} = \begin{pmatrix} m_r & -m_i \\ m_i & m_r \end{pmatrix} * \begin{pmatrix} v_r \\ v_i \end{pmatrix} = \begin{pmatrix} i_r \\ i_i \end{pmatrix} = \underline{i} \tag{4}$$

and then by positive and negative parts through

$$g * v = \begin{pmatrix} g^+ & g^- \\ g^- & g^+ \end{pmatrix} * \begin{pmatrix} v^+ \\ v^- \end{pmatrix} = \begin{pmatrix} i^+ \\ i^- \end{pmatrix}. \tag{5}$$

Eventually, each matrix element in the mathematical kernel yields a 4x4 matrix of memristive conductances and inputs and outputs are expanded into four element vectors following

$$\begin{pmatrix} g_r^+ & g_r^- & g_i^- & g_i^+ \\ g_r^- & g_r^+ & g_i^+ & g_i^- \\ g_i^+ & g_i^- & g_r^+ & g_r^- \\ g_i^- & g_i^+ & g_r^- & g_r^+ \end{pmatrix} * \begin{pmatrix} v_r^+ \\ v_r^- \\ v_i^+ \\ v_i^- \end{pmatrix} = \begin{pmatrix} i_r^+ \\ i_r^- \\ i_i^+ \\ i_i^- \end{pmatrix}. \tag{6}$$

III. Results and Discussion

A. Hardware-aware training for aggressive quantization

In conventional QAT, the scaling factor f_{scale} in equation 3 is specific to each parameter and typically depends on either the maximum or the mean of that parameter. For S4D models, this implies that the real and the imaginary part of the complex matrix $\mathbf{A}$ will have different f_{scale} values, leading to different quantization levels for each part. Consequently, the kernel function in 1 must account for different dynamic ranges of the parameters. In the case of complex multiplication, parameters must be further normalized to ensure that summands with different quantization levels can be properly added. This issue becomes particularly critical when deploying the model on analog computing hardware, where signal dynamic range is typically constrained by supply voltages. Separating signals with different dynamic ranges would require physical separation, along with analog multiplication when the signals are combined. Both of which result in significant power and silicon area overheads.

To address this issue, we introduce quantization-aware training with a fixed dynamic range. This is implemented in the

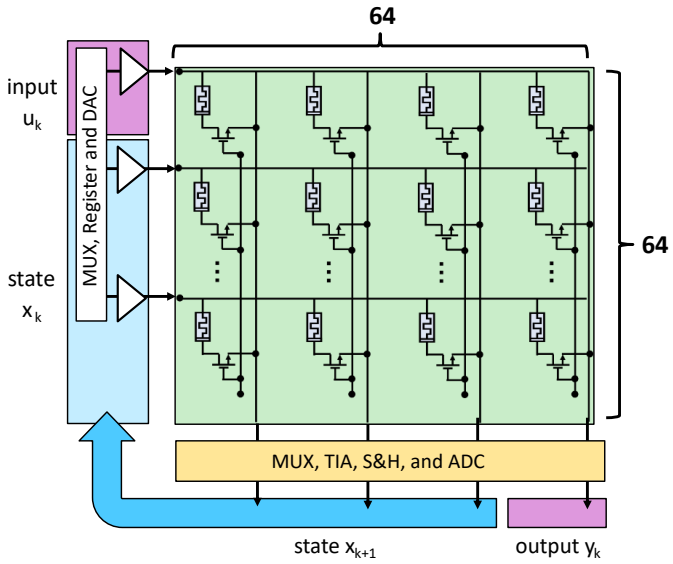

Fig. 1. Memristve State-Space Model Accelerator (IMSSA) kernel implementation on an MCBA.

training algorithm by setting $f_{\text{scale}} = const.$, ensuring that the real and imaginary part of the $\mathbf{A}$ matrix share the same dynamic range. As a result, normalization between real and imaginary computations is no longer required. The impact of this method on model performance for a simple real-world audio classification task is negligible, with the model achieving a classification accuracy of 95.06%, comparable to the model without fixed dynamic range.

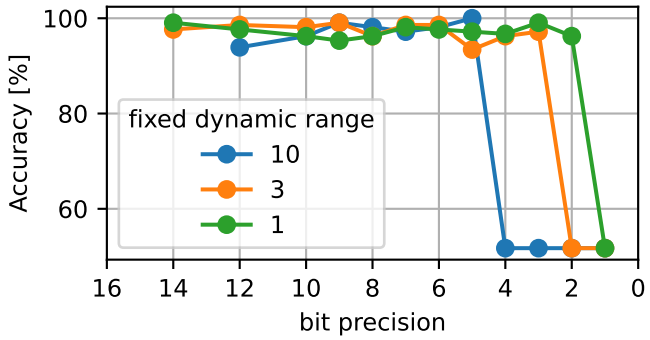

Fig. 2. Training the S4D model with a smaller fixed dynamic range allows for quantization-aware training to lower bit precision without loss of accuracy.

Figure 2 illustrates that various predefined fixed dynamic ranges for the $\mathbf{A}$ matrix can be applied during training. Notably, a smaller fixed dynamic range allows the model to be trained with more aggressive quantization. For instance, with a fixed dynamic range of 1, the model can achieve near-baseline performance even with a 2 bits quantization of $\mathbf{A}$ matrix, which is not feasible without a fixed dynamic range or larger ranges such as 3 or 10. Since quantization normalizes each parameter to a symmetric range around zero, and the real part of the $\mathbf{A}$ matrix is negative or zero while the imaginary part is either positive or zero, a 2 bits quantization effectively reduce the $\mathbf{A}$ matrix to ternary values.

It should be noted that for certain artificial benchmark tasks, such as the sequential CIFAR10 dataset, this method requires higher bit precision compared to using a dynamic and individual f_{scale} value.

B. IMSSA: Realizing the S4D kernel in a single memristive crossbar array

The core computation of S4D models involves the recurrent computation of the state in the kernel $\mathbf{A}x$, while incorporating the input $\mathbf{B}u$ and extracting the output $y = \mathbf{C}x$. These three operations consist of vector or (diagonal) matrix multiplications, meaning that the computational effort scales linearly with the dimension of the state. Instead of treating the three operations separately, which would require physically separating them into different tiles and broadcasting signals between them, we propose to combine all three in a single memristive crossbar array to fully leverage the benefits of in-memory computing. Consequently, the trained kernel matrices $\mathbf{A}$, $\mathbf{C}$, and the (in this case) unit vector $\mathbf{B} = \mathbf{1}$ are programmed into a memristive crossbar array as illustrated in Figure 3. The complex values of these matrices are expanded into memristive conductances using the previously described method. For the diagonal $\mathbf{A}$ matrix, the resulting 4x4 blocks are visible, while the vector $\mathbf{B}$ is arranged horizontally with small diagonals, and the vector $\mathbf{C}$ appears as a vertical block four devices wide. The input signal u is applied in a complex expanded form across the first four rows. In the subsequent rows, a vector of voltage signals representing the state x is applied. At each time step, this vector is obtained by converting the output currents of the MCBA from the first $N * 4$ columns for the next time step, creating a time-delayed recurrent connection. The last four output currents represent the result of the output function $\mathbf{C}x$. As a result, this implementation solves the kernel computation as follows

$$\begin{aligned} x_t &= \overline{\mathbf{A}}x_{t-1} + \overline{\mathbf{B}}u_t \\ y_{t-1} &= \overline{\mathbf{C}}x_{t-1} \end{aligned} \tag{7}$$

executing all operations in an in-memory compute fashion in a single operation. The memristive kernel 7 represents only a slight deviation from the original time-discrete kernel 2, namely a time-shift of the output of one time step.

From this approach for deploying the S4D kernel on an MCBA array, one cane deduce the maximum dimension of the state, which serves as a hyper-parameter for neural network training. The MCBA accelerator chip [8]–[10] we use for deployment features three MCBAs, each with a size of 64x64 memristive devices. Given that four rows and four columns are reserved for the horizontal $\mathbf{B}$ and the vertical $\mathbf{C}$ vector, this leaves a 60x60 sub-array available for potential implementation, in which a 15x15 diagonal matrix $\mathbf{A}$ could be realized. However, for error correction purposes, we opt to use a 14x14 matrix instead. Since we want to deploy a model on a single chip, the number of layers is set to

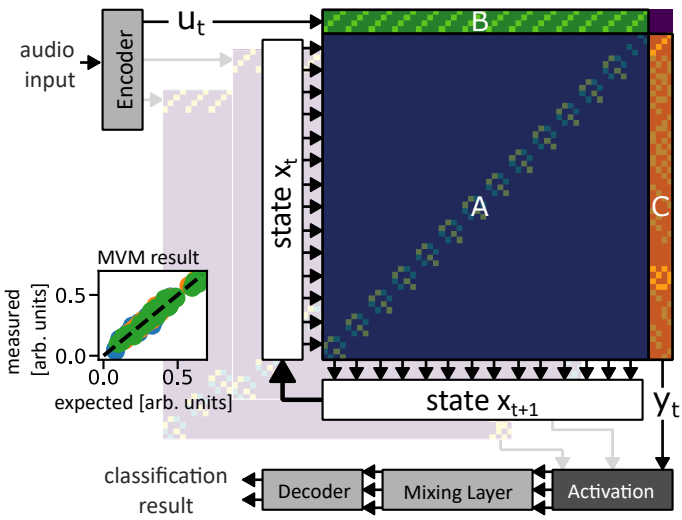

Fig. 3. Integration of the IMSSA architecture in the S4 model. Memristive conductances are color-coded with the respective matrices as overlays.

one , and the number of parallel kernels is limited to three, corresponding to the number of MCBAs per chip. Using the conversion equation 6, and following the QAT process described above, the kernels are written on the MCBAs with memristive conductance values ranging from $7\mu S$ to $200\mu S$. The resulting memristive arrays are shown in Figure 3. With these kernels, the model achieves a classification accuracy of 81.69% on the test data set. While this is significantly above random guessing at 50%, it represents a notable drop compared to the software model's accuracy of 95.06%. As illustrated in Figure 4, this performance is slightly below the distribution of software-calculated accuracies for the write noise level found during programming. The discrepancy can be attributed to faulty devices within the MCBAs, which remain stuck in a high conductive state, causing large incorrect activations. Such issues have been previously reported in the literature for this accelerator platform [9]. Without these extreme programming errors, typically occurring in one to three devices per kernel, the software model accuracy of 95.61% could be fully restored.

To the best of our knowledge, this result presents the first demonstration of an S4D kernel implemented on an analog in-memory compute substrate.

C. Write-noise resilience through quantization

Memristive crossbar architectures today can be realized using various materials and physical mechanisms [14]–[20]. However, most of these approaches suffer from inherent write noise or require complex write schemes to achieve high programming accuracies [21]–[27]. Even though the kernel is typically programmed once during network initialization, a practical use case would also involves reprogramming for updates or newer versions of the network. This imposes constraints on both time and energy budget for programming cycles, leading to some unavoidable level of write noise.

However, strong quantization of the network can mitigate the impact of write noise. To evaluate this, we simulate the deployment of the kernels as described above by injecting Gaussian write noise with zero mean and varying σ into the trained quantized kernels. We then investigate the resulting inference accuracy on the test data set. Figure 4 shows that

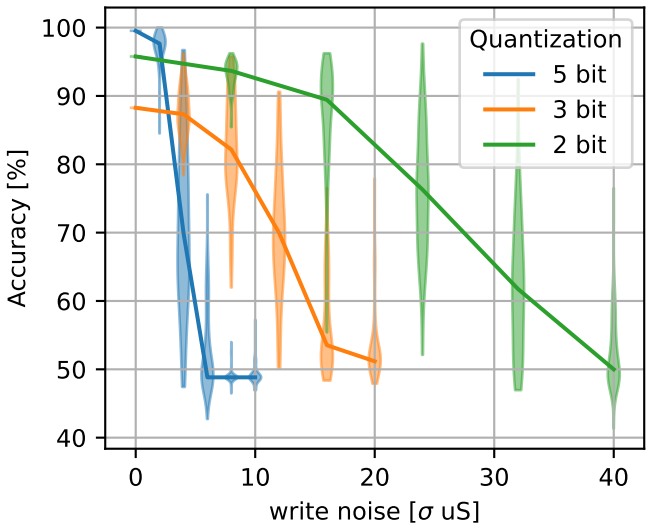

Fig. 4. Influence of write noise on model performance for different kernel quantization. Violin plots indicate the distribution of 100 instantiations around the median.

for a quantization of the kernel parameters to 5 bits already $\sigma = 5\mu S$ lead to a strong decrease in classification accuracy. For a 2 bit quantization, more than $\sigma = 15\mu S$ can be sustained without significantly reducing the performance. This shows how a strong quantization can make a network robust enough to yield high performance on a noisy substrate.

IV. Conclusion

In this work, we demonstrate a comprehensive hardware-aware training process for deploying an S4D model onto an analog in-memory computing substrate. The approach can be generalized to other state-space model architectures like S4 or S5 and other in-memory compute architectures. Using memristive crossbar arrays as an example, we account not only for the limited programming precision of memristive devices, but also for the constrained dynamic range of signals in an analog computing system. By extending a common quantization-aware training method with a fixed dynamic parameter range, we successfully train and deploy an S4D network for an audio classification task. Furthermore, we explore how quantization helps mitigate the effects of programming noise on the network performance. To the best of our knowledge, this is the first dedicated hardware implementation of modern state models using in-memory and analog computing principles, which is an important step toward the efficient deployment of these models in edge computing applications.

REFERENCES

[1] O. Kuchaiev, and B. Ginsburg, "Factorization tricks for LSTM networks," arXiv preprint arXiv:1703.10722, 2017

[2] J. Zhao, F. Huang, J. Lv, Y. Duan, Z. Qin, G. Li, and G. Tian, "Do RNN and LSTM have long memory?," In International Conference on Machine Learning (pp. 11365-11375). PMLR, 2020

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," In Advances in Neural Information Processing Systems, 2017.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," In International Conference on Learning Representations, 2015.

[5] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, ... and D. Metzler, "Long range arena: A benchmark for efficient transformers," arXiv preprint arXiv:2011.04006, 2020

[6] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, "Hippo: Recurrent memory with optimal polynomial projections," Advances in neural information processing systems, 33, 1474-1487, 2020.

[7] S. Abreu, J.E. Pedersen, K.M. Heckel, and A. Pierro, "Q-S5: Towards Quantized State Space Models," arXiv preprint arXiv:2406.09477, 2024

[8] C. Li, J. Ignowski, X. Sheng, R. Wessel, B. Jaffe, J. Ingemi, C. Graves, and J. P. Strachan, "CMOS-integrated nanoscale memristive crossbars for cnn and optimization acceleration," IEEE International Memory Workshop, 2020.

[9] R. Mao, B. Wen, M. Jiang, J. Chen, and C. Li, "Experimentally-validated crossbar model for defect-aware training of neural networks," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, pp. 2468 – 2472, 2022.

[10] R. Mao et al., "Experimentally validated memristive memory augmented neural network with efficient hashing and similarity search," Nat Commun, vol. 13, no. 6284, 2022.

[11] A. Gu, K. Goel, A. Gupta, andC. Ré, "On the parameterization and initialization of diagonal state space models," Advances in Neural Information Processing Systems, 35, 35971-35983, 2022.

[12] A. Gu, and T. Dao. "Mamba: Linear-time sequence modeling with selective state spaces," arXiv preprint arXiv:2312.00752, 2023

[13] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, ... and F. Wei, "The era of 1-bit llms: All large language models are in 1.58 bits," arXiv preprint arXiv:2402.17764, 2024

[14] Rainer Waser and Masakazu Aono. "Nanoionics-based resistive switching memories". Nature Materials 6, 11, 2007.

[15] J. Joshua Yang, Matthew D. Pickett, Xuema Li, Douglas A. A. Ohlberg, Duncan R. Stewart, and R. Stanley Williams. "Memristive switching mechanism for metal/oxide/metal nanodevices". Nature Nanotechnology 3, 7, 2008.

[16] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B. Bhadviya, Pinaki Mazumder, and Wei Lu. "Nanoscale Memristor Device as Synapse in Neuromorphic Systems". Nano Letters 10, 4, 2010

[17] Giacomo Indiveri, Bernabé Linares-Barranco, Robert Legenstein, George Deligeorgis, and Themistoklis Prodromakis. "Integration of nanoscale memristor synapses in neuromorphic computing architectures". Nanotechnology 24, 38, 2013.

[18] Burr, G. W. et al. "Neuromorphic computing using non-volatile memory". Adv Phys X 2, 89–124, 2017.

[19] F. Merrikh Bayat, M. Prezioso, B. Chakrabarti, H. Nili, I. Kataeva, and D. Strukov. "Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits". Nature Communications 9, 1, 2018.

[20] Daniele Ielmini and H.-S. Philip Wong. "In-memory computing with resistive switching devices". Nature Electronics 1, 6, 2018.

[21] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm". Nanotechnology 23, 7, 2012.

[22] Ligang Gao, Pai-Yu Chen, and Shimeng Yu. "Programming Protocol Optimization for Analog Weight Tuning in Resistive Memories". IEEE Electron Device Letters 36, 11, 2015.

[23] Emmanuelle J Merced-Grafals, Noraica Dávila, Ning Ge, R Stanley Williams, and John Paul Strachan. "Repeatable, accurate, and high speed multi-level programming of memristor 1T1R arrays for power efficient analog computing applications". Nanotechnology 27, 36, 2016.

[24] C. Li et al., "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," Nature communications, vol. 9, no. 1, 2018.

[25] V. Milo, A. Glukhov, E. P´erez, C. Zambelli, N. Lepri, M. K. Mahadevaiah, E. P.-B. Quesada, P. Olivo, C. Wenger, and D. Ielmini, "Accurate program/verify schemes of resistive switching memory (rram) for in-memory neural network circuits," IEEE Transactions on Electron Devices, vol. 68, pp. 3832 – 3837, 2021.

[26] M. Rao, H. Tang, J. Wu, W. Song, M. Zhang, W. Yin, Y. Zhuo, F. Kiani, B. Chen, X. Jiang et al., "Thousands of conductance levels in memristors integrated on cmos," Nature, vol. 615, no. 7954, pp. 823–829, 2023.

[27] W. Song et al., "Programming memristor arrays with arbitrarily high precision for analog computing," Science, vol. 383, pp. 903–910, 2024.

[28] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M. Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R. Stanley Williams. "Dot-product engine for neuromorphic computing". Proceedings of the 53rd Annual Design Automation Conference, 2016.

[29] Can Li, Miao Hu, Yunning Li, Hao Jiang, Ning Ge, Eric Montgomery, Jiaming Zhang, Wenhao Song, Noraica Dávila, Catherine E. Graves, Zhiyong Li, John Paul Strachan, Peng Lin, Zhongrui Wang, Mark Barnell, Qing Wu, R. Stanley Williams, J. Joshua Yang, and Qiangfei Xia. "Analogue signal and image processing with large memristor crossbars". Nature Electronics 1, 52-59, 2017.

[30] Hu M, Graves CE, Li C, Li Y, Ge N, Montgomery E, Davila N, Jiang H, Williams RS, Yang JJ, et al. "Memristor-based analog computation and neural network classification with a dot product engine". Adv Mater. 30, 1705914, 2018.

[31] Cai, F., Correll, J. M., Lee, S. H., Lim, Y., Bothra, V., Zhang, Z., et al. "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations". Nat. Electron. 2, 290–299, 2019

[32] H. Zhao, Z. Liu, J. Tang, B. Gao, Q. Qin, J. Li, ... and H. Wu, "Energy-efficient high-fidelity image reconstruction with memristor arrays for medical diagnosis," Nature Communications, 14(1), 2276, 2023