2024 IEEE International Symposium on Circuits and Systems (ISCAS) | 979-8-3503-3099-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISCAS58744.2024.10558167

ReShare: A Resource-Efficient Weight Pattern Sharing Scheme for Memristive DNN Accelerators

Shihao Hong
School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen, China
shihaohong@link.cuhk.edu.cn

Yeh-Ching Chung
School of Data Science
The Chinese University of Hong Kong, Shenzhen, China
ychung@cuhk.edu.cn

***Abstract*—Memristor crossbar-based computing-in-memory (CIM) has garnered significant attention for accelerating deep neural networks (DNNs). Although practical operation unit (OU)-based designs incur abundant repetitive computations, recent studies propose sharing mechanisms, where redundant computations are replaced with data transfers. However, existing sharing approaches carry a heavy burden in terms of storage and energy in practical implementation. To alleviate this problem, this paper proposes a resource-efficient scheme for weight pattern sharing (ReShare). A pattern reorder algorithm is introduced to facilitate the asynchronous running of computations and sharing, which circumvents the bottleneck posed by the output buffer in a conventional sharing scheme. In addition, ReShare enables block write operations to reduce write costs. Our evaluation across four DNN models demonstrates that the proposed ReShare has a lighter overhead in energy and storage while simultaneously enhancing performance. Specifically, ReShare can achieve 1.66× speedup, 30.7% energy saving, and 49.5% storage reduction compared to the state-of-the-art weight sharing method.**

***Index Terms*—Computing-in-memory, memristor, deep neural networks (DNNs), weight pattern sharing (WPS).**

I. Introduction

ReRAM crossbar has emerged as a prominent computing-in-memory (CIM) technology and is widely applied in diverse domains, such as image recognition [1], graph processing [2], DNA alignment [3], Wave Simulation [4], etc. The widespread adoption of ReRAM-based CIM can be attributed to its high computational parallelism and the potential to overcome the memory wall. ReRAM crossbars leverage Kirchoff's Law to perform parallel Matrix-Vector-Multiplication (MVM) directly within the ReRAM cells, all while maintaining the area and power both at a low level. Deep Neural Networks (DNNs), dominated by MVM operations, can benefit greatly from CIM.

However, in the real world, the practical ReRAM-based DNN accelerators grapple with challenges in ReRAM storage and detracted parallelism. When solely utilizing ReRAM cells to store an entire DNN model, the requisition for ReRAM devices becomes significant. For example, the full-precision VGG16 necessitates in excess of 66,000 single-level 128×128 crossbar arrays. With the expanding dimensions of DNN models, this demand amplifies, leading to marked challenges concerning ReRAM storage capacity. Concurrently, the pursuit of high computational accuracy necessitates a reduction in parallelism. Activating excessive ReRAM cells simultaneously makes analog computation suffer from fluctuation due to hardware variation [5]. Many studies [6]–[8] adopt Operation Unit (OU) for better fault resilience, activating only H_{OU} wordlines and W_{OU} bitlines per cycle. Nonetheless, this resilient computation paradigm separates a complete MVM into $\frac{H_M}{H_{OU}}\frac{W_M}{W_{OU}}$ sub-MVMs, curtailing the inherent parallelism.

Many pruning methods have been proposed for ReRAM-based accelerators, which leverage the inherent sparsity [6], [9]–[13] and repetitiveness [14] of weights to reduce the demand for ReRAM crossbars and shorten computation time. CPR [11] densifies the subarray, pruning the model at crossbar-grain. SRE [6] compresses weights at OU-row or OU-column granularity. The approach in [12] clusters and prunes the zeros in weights at OU-grain. APQ [13] finds a global optimal pruning policy for weights in DNN models at the OU-column level. Nonetheless, the above OU-based pruning methods solely utilize weight sparsity but ignore the remaining information in the model. Repetitiveness in non-zero weights also holds the potential for compression. PattPIM [15] records repeated OU patterns in weights and introduces an OU-level repetition pruning method. RePIM [14] delves into the repetition of OU-columns and uses an indexing table to share the computation results. Further, PRAP-PIM [16] fine-tunes the model to increase the ratio of repetition. The above methods require index tables for the dispersion of computational results. However, implementing index tables necessitates a task list recording write addresses and a length list to provide the length of each task. Moreover, the discontinuity in the destination addresses of the disseminated write operations introduces the under-utilization of block data transfer in write operations. Furthermore, mutual dependency between computation and sharing arises when a buffer serves dual roles as both the write target for computational results and the read target for distribution. It implies that an intensive distribution task can obstruct the following computation on hold.

In this paper, we propose ReShare to avoid the mutual dependency problem and exploit continuous writes in the sharing process. Our contributions are as follows:

(1) We discuss the overhead and performance of the weight sharing mechanism in recent works. Key challenges include the absence of block write support, the output buffer acting as a bottleneck, and significant storage costs of auxiliary lists. We conduct preliminary experiments

revealing a dense distribution of weight patterns with the weight matrix, further enhancing our discussion.

(2) We propose ReShare, which consists of a pattern reordering algorithm to enable asynchronous running and a block write-aware architecture design. In architecture, we propose maintaining a task list containing the pattern members of result blocks to facilitate block data transfer.

(3) We evaluate ReShare's performance with popular DNN models on the Imagenet dataset. ReShare achieves at most 1.66× speedups than the ORC method in RePIM. In the metric of energy consumption and storage overhead, ReShare also outperforms the baseline methods.

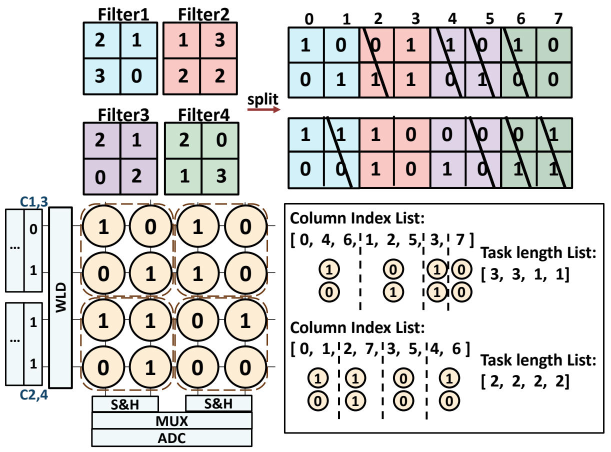

Fig. 1. Pruning repetitive weights in OU-grain and a practical design of the index table. Here, the Weights of the four filters are split and combined into single-bit matrices.

II. Background and Motivation

Pruning repetitive weights and sharing MVM results of weight patterns effectively reduces the ReRAM devices and redundant computations [14]–[16]. A weight pattern is an OU-column vector in a weight matrix. We set the OU in Fig. 1.'s example as a 2×2 square, so there are four distinct weight patterns, i.e., {0,0}, {0,1}, {1,0}, and {1,1}. The duplication pruning strategy first maps the reduced weight matrices onto crossbar arrays and stores key-value pairs of pattern and index:{{1,0}:{0,4,6}, {0,1}:{1,2,5}, {1,1}:{3}, {0,0}:{7}}. The first MVM produces the results for patterns {1,0} and {0,1} and writes to the output buffer. The sharing process forwards the pattern results to the result buffer with respective target addresses, i.e., {0,4,6} and {1,2,5}. Cutting off duplicate computations saves 50% of computations and ReRAM cells in this example, but it also reveals three problems: (1) heavy write overhead, (2) inter-dependency between computation and sharing, and (3) overhead of auxiliary lists.

The target addresses for result distribution are not contiguous because each weight pattern potentially appears in various columns of the weight matrix, and the storage of results is consistent with the column order of the weight matrix. The result buffer has a minimal unit for write operations, which is larger than a single pattern result. However, the sharing

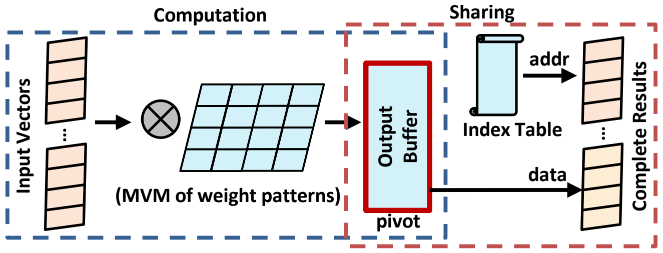

Fig. 2. The flow chart of traditional weight pattern sharing scheme.

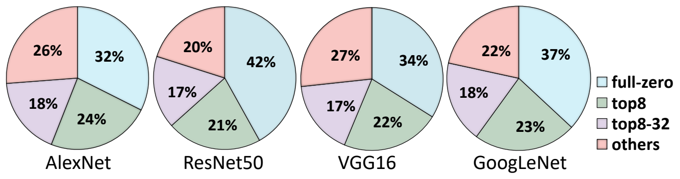

Fig. 3. The occurring frequency of weight patterns in the four models.

scheme misses this advantage due to discontinuous writing and induces a heavy writing burden.

Fig. 2. illustrates that the output buffer acts as a pivot, functioning as the write target for MVM operations and the originating source for sharing. The computation will be blocked when the sharing is under a heavy burden. Such inter-dependency turns the output buffer into a performance bottleneck, increasing latency overhead, especially with busy patterns. Alternatively, the design shall enlarge the output buffer to enable asynchronous running between computation and sharing, increasing the space overhead.

For the index table design, employing a fixed-size block for each pattern to record target addresses leads to substantial space wastage for infrequent patterns. This wastage is dictated by the uneven distribution of patterns in the weight matrix, coupled with the block size determined by the largest task list. A practical implementation of the index table consists of a column index list and a task length list. As shown in Fig. 1., once an MVM is completed, the pattern first accesses the task length list to ascertain the task amount. Subsequently, it navigates to the column index list to obtain respective target addresses. The spatial complexity associated with the two auxiliary lists is $O(H_M W_M log(W_M))$ and $O(H_M N_p log(W_M))$, where H_M, W_M, N_p are height and width of the weight matrix, and the number of patterns, respectively.

We conduct pre-experiments to investigate the distribution of weight patterns in the models (i.e., AlexNet, ResNet50, VGG16, and GoogLeNet) where OU is set as a 8×8 square. As shown in Fig. 3., all four models exhibit that more than 21% and 38% of OU-columns are dominated by the most frequently occurring eight (top-8) patterns and top-32 weight patterns. The numbers will increase to 33% and 59% when only non-zero patterns are considered. The observed concentration of patterns enhances the previously discussed spatial inefficiencies and reaffirms the output buffer's role as a pivotal bottleneck, obstructing subsequent computations. We are then motivated to propose ReShare, a sharing scheme supporting asynchronous running and block writing.

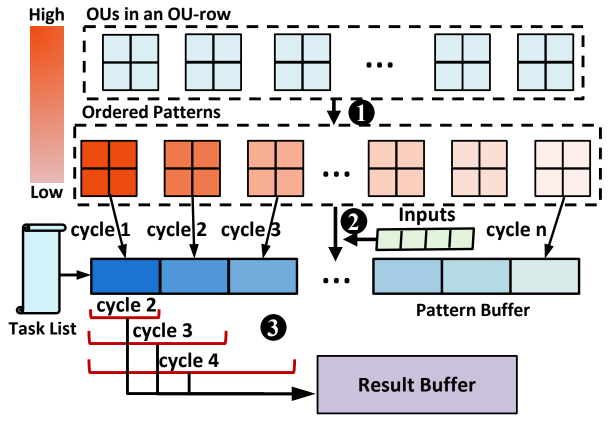

Fig. 4. Data flow of ReShare. (1) Order patterns; (2) Compute pattern results; (3) Distribute from pattern buffer to result buffer;

III. Proposed Weight Pattern Sharing Scheme

A. Data Flow

Fig. 4 shows the idea and data flow of the proposed ReShare. 1) An offline ordering process is applied to weight patterns according to their capability of constructing complete blocks. 2) MVM units perform computation on the ordered patterns and forward results to the pattern buffer in the ordered sequence. 3) Pattern Buffer (PB) receives distribution tasks from the task list and writes goal data blocks to Result Buffer. Race conditions are avoided in ReShare because the blocks of PB that originate the sharing data were written in prior cycles and will not be overwritten by subsequent pattern results.

B. Pattern Reordering Algorithm

ReShare proposes an algorithm for reordering weight patterns, detailed in Algorithm 1, which prioritizes the computation of patterns that play a more significant role in forming complete blocks. We define the value of a pattern as the summation of weighted frequencies in all blocks (Line 9):

$$\text{Value}(x) = \sum_{\text{block}} \beta \cdot \text{freq}(x, \text{block}), \quad \text{where} \quad \beta = 1/\text{count}(\text{candidate patterns in block}) \tag{1}$$

The weight coefficient β is calculated as the reciprocal of the number of candidate patterns appearing in the current block (Line 6). And freq(x, block) returns the showing times of pattern x in the block. After iterating all blocks, the pattern in the unselected list with the highest value will join the selected pattern list (Line 10,11). Each iteration would recalculate the pattern value. The proposed weighted design ensures that patterns contributing significantly to block write operations are accorded higher precedence in ranking.

C. Architecture Design

Fig. 5. illustrates the architecture for ReShare. It is composed of a global controller generating control signals to

Algorithm 1: Pattern Reordering Algorithm

```
Input  : OU-column set $S_{col}$, #Pattern in $S_{col}$ $N$
Output: Reordered weight patterns $P_{order}$
1  Init dynamic value list $L_{val}$ as all-zero, selected
     pattern list $L_{sel}$ as empty.
2  for $i = 0$ to $N$ do
3      for each non-zero block in $S_{col}$ do
4          if block $\not\subseteq L_{sel}$ then
5              Count unselected patterns in block $n_{unsel}$
6              $coeff \leftarrow 1/n_{unsel}$
7              for col in block do
8                  if $col \neq 0$ and $col \notin L_{sel}$ then
9                      $L_{val}[col]$ += $coeff$
10     $Patt_{max} \leftarrow argmax(L_{val})$
11     Append $Patt_{max}$ to $L_{sel}$
12 /* $L_{sel}$ is ordered by patterns'
       contribution, so $L_{sel}$ is $P_{order}$.     */
13 return $L_{sel}$
```

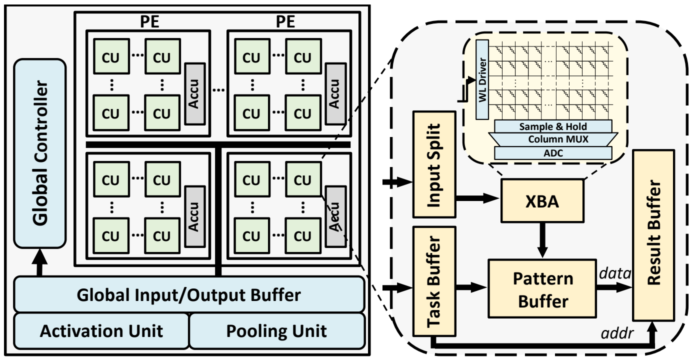

Fig. 5. Hierarchical illustration of the architecture design.

all components, a global buffer storing input feature maps and output data, activation units and pooling units handling intermediate outputs, and multiple Processing Engines (PEs) consisting of multiple Compute Units (CUs) conducting computation and sharing. The input dispatcher and task buffer receive signals from the controller. The shared pattern results are written to the result buffer and are accumulated by the shared accumulators. The pattern buffer and task buffer are the fundamentals of ReShare's sharing scheme. The task buffer records each pattern result's task details, including the block index (write address to result buffer) and pattern index (read address of pattern buffer). Since the block is fixed-size globally, accessing the task buffer doesn't need the auxiliary lists we discussed previously. The specific cost reduction will be presented in the Evaluation section.

IV. Evaluation

A. Experimental Setup

Based on MNSIM [17], we implement a prototype of ReShare. The latency delay of the eDRAM buffers, such as Pattern Buffer and Result Buffer, are modeled using CACTI

TABLE I
HARDWARE CONFIGURATION

CU configuration (16 CUs per PE)		
Memristor Array	number: 8; size: 32×128 bits-per-cell: 1 bit; OU-size: 8×8	0.51mW
DAC	Number: 8×32; Resolution: 1 bit	1.0 mW
ADC	Number: 8; Resolution: 6 bit Frequency: 1 GSps	12.1 mW
S+A	Number: 4	0.2mW
S+H	8×128	10 uW
eDRAM Buffer	size: 32KB	14.3 mW

[18] at the 32-nm process. The configuration is set to meet the requirements of four benchmark DNN models: Alexnet, GoogleNet, ResNet50, and VGG16. Table I summarizes the hardware configuration of the architecture. Each PE has 16 CUs, and each CU groups 8 Single-Level-Cell (SLC) ReRAM crossbar arrays. The result buffer, pattern buffer, and task buffer are assembled in the on-chip buffer of each CU.

Tasks and Baselines: We choose the ImageNet [19] classification task as the benchmark, involving a large volume of intermediate OU-input data. We select four representative DNN models, i.e., Alexnet, GoogleNet, ResNet50, and VGG16, to evaluate the performance. Weights are quantized to 8-bit using a post-training quantization algorithm [20]. All evaluations adopt an 8×8 OU design. The block size for memory write is set to 16B. We compare the performance of ReShare against two state-of-the-art weight pruning approaches, i.e., OU-row compression (ORC) in SRE [6] and repetitive weight sharing (RWS) in RePIM [14], on four metrics: latency, ReRAM compression rate, energy consumption, and storage overhead. All the results are normalized to the ORC method or RWS.

B. Experimental Results

1) Speedup: Fig. 6 (a) presents the execution time of ReShare and baseline methods. ReShare achieves at most 1.66× speedup compared with RWS. The crossbar array configuration with fewer rows increases the MVM parallelism for OU-based computation, contributing to the performance. This improvement also illustrates that leveraging block data transfers, combined with asynchronous computation and sharing, effectively conceals transfer duration behind computation time.

2) ReRAM reduction: Considering that RePIM utilizes crossbar arrays to store both weights and the indexing table, we assess the ReRAM requirement based solely on weights to ensure a fair comparison. Fig. 6 (b) shows that ReShare and RWS have almost the same demand on ReRAM cells because they only perform the MVMs on limited weight patterns. In other words, the ReRAM demand does not vary with the matrix width. ReShare exhibits a superior compression rate on AlexNet, ResNet50, and VGG16 in comparison to GoogLeNet. This is attributed to the layers in GoogLeNet having fewer columns than the others, leading to suboptimal utilization of the sharing scheme.

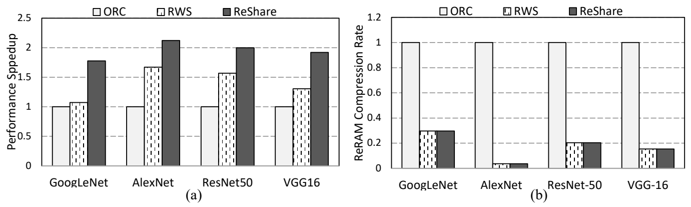

Fig. 6. Performance speedup and ReRAM device reduction.

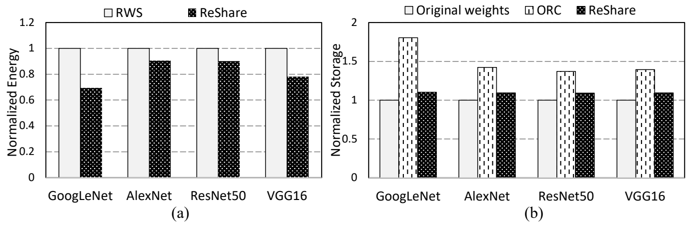

Fig. 7. Normalized energy and storage overhead on different models.

3) Energy saving: The energy consumption of ReShare normalized to RWS is shown in Fig. 7(a). Though RWS adopts low-power ReRAM devices to store the indexing table, ReShare saves at most 30.7% of energy in comparison with RWS, which demonstrates the benefit of block data transfer. Nonetheless, when the weight matrix is wide, e.g., AlexNet, the sharing process requires longer execution time and more energy consumption.

4) Storage overhead: The main storage cost is comprised of the result buffer, pattern buffer, and task buffer. Fig. 7(b) illustrates the normalized storage overhead of the four models. It shows that, in practice, ORC needs more storage than original weights, especially when the sparsity in the weight matrix is rather non-ideal due to the quantization algorithm. In summary, ReShare decreases the storage overhead by 49.5% in comparison with ORC.

V. CONCLUSION

This paper discusses the performance bottleneck and storage overhead inherent in recent repetitive weight sharing mechanisms for memristive DNN, highlighting the substantial costs of execution time, storage, and energy consumption associated with the sharing process in practical implementation. We propose ReShare to mitigate this problem, including a pattern reordering algorithm and a tailored architecture. We discover the computation flow supporting asynchronous execution and block data transfers, reducing overall overhead with minor hardware costs. Evaluations show that ReShare achieves at most 1.66× speedups over the representative weight sharing method while improving storage and energy efficiency.

VI. ACKNOWLEDGMENT

This work is partially supported by Huawei Technologies Co., Ltd under contract No.TC20220913039.

References

[1] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[2] S. A. Ghasemi, B. Jahannia, and H. Farbeh, "Grapha: An efficient reram-based architecture to accelerate large scale graph processing," *Journal of Systems Architecture*, vol. 133, p. 102755, 2022.

[3] W. Huangfu, S. Li, X. Hu, and Y. Xie, "Radar: A 3d-reram based dna alignment accelerator architecture," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[4] B. Hanindhito, R. Li, D. Gourounas, A. Fathi, K. Govil, D. Trenev, A. Gerstlauer, and L. John, "Wave-pim: Accelerating wave simulation using processing-in-memory," in *Proceedings of the 50th International Conference on Parallel Processing (ICPP)*, 2021, pp. 1–11.

[5] M.-Y. Lin, H.-Y. Cheng, W.-T. Lin, T.-H. Yang, I.-C. Tseng, C.-L. Yang, H.-W. Hu, H.-S. Chang, H.-P. Li, and M.-F. Chang, "Dl-rsim: A simulation framework to enable reliable reram-based accelerators for deep learning," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[6] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*, 2019, pp. 236–249.

[7] C.-Y. Wang, Y.-W. Chang, and Y.-H. Chang, "Sgirr: Sparse graph index remapping for reram crossbar operation unit and power optimization," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–7.

[8] Y.-L. Zheng, W.-Y. Yang, Y.-S. Chen, and D.-H. Han, "An energy-efficient inference engine for a configurable reram-based neural network accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 3, pp. 740–753, 2022.

[9] F. Liu, Z. Wang, Y. Chen, Z. He, T. Yang, X. Liang, and L. Jiang, "Sobs-x: Squeeze-out bit sparsity for reram-crossbar-based neural network accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 204–217, 2022.

[10] F. Liu, W. Zhao, Z. He, Z. Wang, Y. Zhao, Y. Chen, and L. Jiang, "Bit-transformer: Transforming bit-level sparsity into higher preformance in reram-based accelerator," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[11] J. Park and S. Kang, "Cpr: Crossbar-grain pruning for an rram-based accelerator with coordinate-based weight mapping," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 336–343.

[12] H. Shin, R. Park, S. Y. Lee, Y. Park, H. Lee, and J. W. Lee, "Effective zero compression on reram-based sparse dnn accelerators," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 949–954.

[13] S. Yang, S. He, H. Duan, W. Chen, X. Zhang, T. Wu, and Y. Yin, "Apq: Automated dnn pruning and quantization for reram-based accelerators," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[14] C.-Y. Tsai, C.-F. Nien, T.-C. Yu, H.-Y. Yeh, and H.-Y. Cheng, "Repim: Joint exploitation of activation and weight repetitions for in-reram dnn acceleration," in *58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 589–594.

[15] Y. Zhang, Z. Jia, Y. Pan, H. Du, Z. Shen, M. Zhao, and Z. Shao, "Pattpim: A practical reram-based dnn accelerator by reusing weight pattern repetitions," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[16] Z. Shen, J. Wu, X. Jiang, Y. Zhang, L. Ju, and Z. Jia, "Prap-pim: A weight pattern reusing aware pruning method for reram-based pim dnn accelerators," *High-Confidence Computing*, vol. 3, no. 2, p. 100123, 2023.

[17] L. Xia, B. Li, T. Tang, P. Gu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "Mnsim: Simulation platform for memristor-based neuromorphic computing system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 1009–1022, 2017.

[18] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[19] 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*, 2009, pp. 248–255.

[20] J. Zhang, Y. Zhou, and R. Saab, "Post-training quantization for neural networks with provable guarantees," *SIAM Journal on Mathematics of Data Science*, vol. 5, no. 2, pp. 373–399, 2023.