AutoGMap: Learning to Map Large-Scale Sparse Graphs on Memristive Crossbars

Bo Lyu, Shengbo Wang, Shiping Wen, Senior Member, IEEE, Kaibo Shi, Member, IEEE, Yin Yang, Lingfang Zeng, and Tingwen Huang, Fellow, IEEE

Abstract—The sparse representation of graphs has shown great potential for accelerating the computation of graph applications (e.g., social networks and knowledge graphs) on traditional computing architectures (CPU, GPU, or TPU). But, the exploration of large-scale sparse graph computing on processing-in-memory (PIM) platforms (typically with memristive crossbars) is still in its infancy. To implement the computation or storage of large-scale or batch graphs on memristive crossbars, a natural assumption is that a large-scale crossbar is demanded, but with low utilization. Some recent works question this assumption; to avoid the waste of storage and computational resource, the fixed-size or progressively scheduled “block partition” schemes are proposed. However, these methods are coarse-grained or static and are not effectively sparsity-aware. This work proposes the dynamic sparsity-aware mapping scheme generating method that models the problem with a sequential decision-making model, and optimizes it by reinforcement learning (RL) algorithm (REINFORCE). Our generating model [long short-term memory (LSTM), combined with the dynamic-fill scheme] generates remarkable mapping performance on the small-scale graph/matrix data (complete mapping costs 43% area of the original matrix) and two large-scale matrix data (costing 22.5% area on qh882 and 17.1% area on qh1484). Our method may be extended to sparse graph computing on other PIM architectures, not limited to the memristive device-based platforms.

Index Terms—Large-scale graph, long short-term memory (LSTM), memristor, reinforcement learning (RL), sparsity.

I. INTRODUCTION

GRAPH data structure is typically represented by an adjacency matrix, which has extensive sparsity [1], [2], [3], [4]. It relies on traditional matrix compression format, e.g., CSR, CSC, and COO, to save substantial storage resources and is effectively processed by the sparse-based computing algorithm or library (SpMM and SpMV) on traditional computing devices (CPU, GPU, or TPU) [5]. However, in terms of processing-in-memory (PIM) or computation-in-memory (CIM) platforms, which are typically implemented by memristive crossbars, the graph data must be restored from the storage format to the computing format (adjacency matrix) [1] before mapping. Directly mapping large-scale sparse graph data on a crossbar is not appropriate, for it will seriously affect the utilization and power consumption of memristive platforms. In addition to the large-scale graph data, it is also fatal for batch graphs computing, in which the adjacency matrices are usually integrated into a large-scale supermatrix, with only the subgraphs being internally connected, and the adjacency relationship across the graphs are null [6]. Unfortunately, the current fabrication technology of memristor crossbars is immature, and it is difficult to fabricate large-scale memristive crossbars with high yield. Therefore, it is necessary to make efficient usage of the discrete small-scale crossbars on memristor-based computing platforms.

In recent years, some PIM-based studies [1], [2], [7] have made effective attempts in the efficient computation of sparse graphs on neuromorphic computing platforms. Following clustering or reordering [8] the sparse adjacency matrix, some fixed-size block partition or progressive partition schemes are proposed, among which only the blocks with nonzero entries need to be mapped to the crossbar [1], [2], [6]. From our perspective, these schemes fail to propose an effective mapping scheme for the graph data, which should be feasible, scalable, dynamic, and flexible, especially in the scenario of large-scale graph or batch graphs computing. This article focuses on filling this gap, which is crucial in optimizing the computing/storage efficiency and resource utilization of large-scale graphs on the memristive platforms. According to the scenarios and characteristics of memristive crossbar-based computing, we first propose several principles of dynamic sparsity-aware mapping and coding framework. We then model the problem as a sequential decision-making problem, which is heuristically sampled and optimized by a
policy gradient-based reinforcement learning (RL) algorithm. Eventually, our experiments show remarkable mapping results on both small-scale and large-scale graph/matrix datasets.

Overall, our contributions can be summarized as follows.

1) **Modeling:** We simplify the mapping scheme generation problem and further formulate it as the sequential decision-making problem.

2) **Optimization:** We further exploit long short-term memory (LSTM) + dynamic fill to model the problem and optimize it heuristically by the RL algorithm.

3) **Scalability:** Our method can meet the real constraint (allowable limited crossbar size, the complexity of peripheral circuits, and so on) of the deployed platforms; that is, it is flexible and scalable.

Our work eliminates the dependence of large-scale graph computing on integrated crossbars and allows for the utilization of discrete crossbars with limited sizes. This improves the utilization and feasibility of memristive crossbars, enabling dynamic sparsity-aware mapping in large-scale graph computing. Our code is available at https://github.com/blyucs/AutoGMap.

II. RELATED WORK

A. **PIM and Neuromorphic Computing**

Data exchange between PUs and off-chip storage devices (hard drives and flashes) consumes two orders of magnitude more energy than a floating-point operation [9]. It becomes more serious for neural network (NN) applications and real-time automatic systems [10], [11], [12], [13], which substantially rely on data storage and memory exchange (both with respect to feature maps and weight parameters). The “memory wall” problem in von-Neumann architecture will surely become a bottleneck for progress in these areas. Given the capacity to achieve low-power consumption and low inference latency, PIM [14], [15], [16] is a feasible solution to tackle these issues [17], [18], which integrates the computation and data storage.

B. **Memristor Crossbar-Based Computation**

The emerging studies of memristor [19], [20], [21], [22] have shown its great potential on PIM platforms. With the memristors structured into the crossbar, it performs matrix-vector multiplication efficiently and has been widely studied to accelerate NN applications. Much more works have studied the deployment of different types of NNs (ex situ) with respect to different benchmark tasks on memristor crossbar arrays [23], [24], [25]. Other works also study the training procedure of NNs (in situ) [26], [27], [28], [29], [30], [31]. Wen et al. [23] propose a novel memristor-based computational architecture for the echo state network (MESN) with the online least mean square (LMS) algorithm. Chen et al. [32] design the memristor-based circuit to implement fully convolutional networks (FCNs) for image segmentation application. Recently, as a newly evolutionary NN structure, the Transformer [33] has also been studied under the scenario of memristor crossbar arrays [25]. Our work is also partially motivated by [32] and [34], which are concerned with the acceleration of the hardware implementation of artificial NNs.

C. **Resource-Aware Computation and Energy Saving**

Recently, much more attention has been paid to the resources and energy consumption of computing. Some notable works on resource-aware computation optimization, and software/hardware co-design models [35], [36] are proposed. Targeting the resource consumption and energy saving on neuromorphic computing platforms, some works [37], [38] propose some effective ways for accelerating the memristor-based CNNs on classification or segmentation tasks.

D. **Graph Data Processing and Acceleration**

Toward the sparse matrix-vector multiplication based on memristor crossbar, Cui et al. [7] propose an improved generalized reordering algorithm based on Cuthill–McKee reordering to reduce the bandwidth of graph adjacency matrix, thus improving the efficiency of the crossbars, whereas no blocks mapping scheme is studied after reordering. Balog et al. [6] directly utilize the Cuthill–McKee reordering algorithm followed by three densified diagonal-blocks coverage, to speed up the training efficiency, thus demonstrating the competitive efficiency on TPUv2 (dense acceleration hardware). Targeting the heterogeneous accelerators for graph computation on PIM architecture, some works [1], [2] propose sparse graph partition schemes to improve the storage efficiency. GraphR [1] statically partitions adjacency matrix and combines sparse compression format, to reduce memory consumption. On the WikiNote dataset, it cost only 0.2% of the original size when combined with the COO representation. GraphSAR [2] proposes the sparsity-aware partition scheme to store the large-scale sparse matrix. After partitioning the adjacency matrix into fixed small size (e.g., 8 × 8), it only directly stores the blocks with a nonzero density larger than 0.5, otherwise further divides the blocks into 4 × 4. By this means, the large-scale graphs data are stored in the form of small matrices, thus greatly increasing the utilization of the discrete fragmentary crossbars. Although these works endeavor to improve the computation efficiency and resource utilization by block partition scheme, no dynamic and real sparsity-aware partition methods are proposed.

III. PRELIMINARY

In parallel computing on conventional hardware (CPU and GPU), the Intel MKL, NVIDIA cuBLAS library, cuSparse, and other libraries directly implement SpMV based on the compressed format. However, for large-scale graph data or batch graphs, although the memristor-based crossbar operation utilizes a process-in-memory architecture to reduce the complexity of matrix-vector multiplication from \(O(n^2)\) to \(O(1)\), it is clear that the complete mapping of the integrated matrix to the crossbar is unwise. Taking spectral-based GCN [39] as an example, the layerwise propagation is given as follows:

\[
Z_{l+1} = \sigma (\hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2} Z_l W_l).
\]  

(1)

It involves the whole sparse adjacency matrix, so complete mapping of graphs on crossbar/hypercube is very resource-consuming. A preferable way is to divide the sparse matrix...
into blocks, and only the blocks containing nonzero elements need to be mapped [6], [7]. However, the nonzero elements of the sparse matrix with graph structure are generally scattered, which will increase the complexity of peripheral circuits and communication between subcrossbars (blocks) [7]. Targeting the goal of “communication optimal” [7], the communication of the blocks in the same row needs to be minimized, thus reducing the complexity of the peripheral circuit. To achieve this, nonzeros need to be located closer enough and preferably distributed around with the diagonal.

Matrix-vector multiplication is the common atomic operation of current artificial intelligence applications (e.g., CNNs [40], [41], [42], RNNs [43], and GNNs [39], [44], [45]) and consists of the multiplication and accumulation. In the computation of the memristive crossbar, multiplication is implemented by Kirchhoff’s current law in analog domain [46]. Fig. 1 showcases the circuit simulation process of a matrix-vector multiplication propagation (\( y = Ax \)) by Kirchhoff’s current law in analog domain [46]. Fig. 1 (right) shows the circuit for multiplication propagation \((y = Ax)\) by Kirchhoff’s current law in analog domain [46].

To ensure the calculation rules of block matrix multiplication, the input vector needs the transformation, denoted as follows:

\[
\text{If } y = Ax, \text{ then } \text{by Cuthill–Mckee reordering, the matrix } A \text{ is transformed to } A' = P A P^T. \tag{3}
\]

which is feasible to be deployed on the crossbar with fewer cost [7]. To ensure the calculation rules of block matrix multiplication, the input vector needs the transformation, denoted as follows:

\[
\text{If } y = Ax, \text{ then } \text{by Cuthill–Mckee reordering, the matrix } A \text{ is transformed to } A' = P A P^T. \tag{3}
\]

The calculation after the transformation is represented as follows:

\[
y' = A' x' = P A P^T P x = P A x = P y. \tag{5}
\]

Then, after the calculation of crossbars, the composite output vector \( y' \) needs to be reversely transformed back to \( y \), denoted as follows:

\[
y = P^T y'. \tag{6}
\]

which may be realized by the switch circuit during the hardware design.

More acceleration and deployment background are proposed in [47]. Considering only diagonally connected block partition, as shown in Fig. 2, different block schedule schemes come up with different coverage ratios and total area (cost). Therefore, the question worth exploring is how to generate the best mapping scheme (complete coverage) with minimum cost.

### IV. PROBLEM FORMULATION

For simplicity, we first consider the formulation of the mapping scheme with only diagonal blocks. The optimization objective is to maximize the expectation of the reward function of the candidate scheme, as follows:

\[
\text{argmax}_s \ f(s_1, s_2, s_3, \ldots, s_n)
\]

\[
\text{s.t. } \sum_{n=1}^{n} s_n = N \\
N \leq \text{Area}
\]

where \( f \) is the reward function of one candidate mapping scheme, \( s \) is the “size” vector of the sequential blocks, and \( N \) is the diagonal size, as illustrated in Fig. 3 (left). In terms of the solution space, this problem can be analogous to a typical pure numerical problem, the integer factorization by addition; that is, given a positive integer \( n \), the number of the factorization schemes is \( 2^{N-1} \). So, the algorithm complexity of the violent solution is \( O(2^N) \). Considering a large numerical value \( N \) (e.g., 100 and 1000), solving this problem violently is obviously computationally impossible. In addition, it is infeasible to formulate this problem as the integer programming or the discrete optimization problem, for the variable number is uncertain.

We endeavor to transform the expression of the solution space; instead of using the size variables with an uncertain number, we resort to \( N-1 \) variables \((0/1)\) to form the solution space. This may be detailed interpreted as that there exist \( N-1 \) decision points on the diagonal of the adjacency matrix, and the action space of each point is 0: start a new block and 1: continued to expand the previous block, represented by \( x_i \). Then, the optimization problem is transformed to the

Fig. 1. Diagram of matrix-vector multiplication propagation \((y = Ax)\) with the matrix reordering method. The matrix \( A' \) is programmed into a batch of small-scale crossbars, and the transformed vector \( x' \) serves as the inputs of the crossbars. Finally, the switch circuit is resorted to realize the reverse transformation \( y = P^T y' \).

Fig. 2. Comparison of coverage ratio and blocks area (cost) under different mapping schemes. To realize the complete mapping (coverage), the mapping blocks generally cost much more area, as the left scheme shows. But, the middle and right schemes are infeasible for the deployment, which fails to reach the complete coverage.
following:

\[
\begin{align*}
\arg\max_x & \quad f(p(x_1, x_2, x_3, \ldots, x_N)) \\
\text{s.t.} & \quad x_i \in \{0, 1\}
\end{align*}
\]

(8)

where \( p \) is the parse function that transfers 0-1 decision variable \( x \) to the block size vector \( s \), as illustrated in Fig. 3 (right). Although the size of the solution space is the same with (7), \( 2^{N-1} \), consistent with the 0-1 nonlinear integer programming problem, it is easier to formulate this problem as a programming problem. Unfortunately, it still cannot be solved by optimization methods, for the definition of the optimization objective is not a continuous high-dimensional hyperplane, but regarding the distribution of nonzero elements of the matrix, which is in a discrete space. Based on the background of the crossbar and the formulated problem, we put forward the basic principle of mapping large-scale sparse graphs on memristive crossbars.

1) Complete coverage capability, valid for arbitrary nonzeros distribution, and do not exceed the whole area.
2) No overlaps between blocks.
3) Adaptable to the deployment and compile system, that means, coding rules should be simple for the circuit design.
4) Least cost of crossbar area.

V. METHODOLOGY

A. Modeling

1) Modeling Diagonal Blocks: Based on the problem formulation and the background of crossbar-based computation (Fig. 1), optimization variables defined in (8) are pre-and-post related, and we may treat it as a sequential generation problem and further leverage the classical LSTM networks as the sampling controller. For each time step (\( t \) stands for the decision points number), the output embedding data model the representation of each action and serve as the input vector of the next time step, and the propagation of LSTM cell calculates as follows:

\[
\begin{align*}
\sigma(f_t) &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
\sigma(i_t) &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
\tanh(g_t) &= \tanh(W_g[h_{t-1}, x_t] + b_g)
\end{align*}
\]

(9)(10)(11)

where \( W \) denotes the weights and \( b \) represents the bias. At time step \( t \), \( f_t, i_t, g_t \), and \( o_t \) represent the forget gate, input gate, cell state, and output gate, respectively. \( h_{t-1} \) and \( x_t \) signify the hidden state at time \( t-1 \) and the input at time \( t \), respectively. The sequential generation technique can be formulated as a classification problem and accomplished by a fully connected network that takes the LSTM cell’s output embedding vector (per time step) as input

\[
p' = \text{softmax}(W'_f \ast \text{out}_t + b').
\]

(15)

The output indices that represent the index/operations are sampled according to the multinomial distribution \( (p'_1, \ldots, p'_C) \), where \( C \) is the classification number of FC. The sampling details are presented in Algorithm 1.

2) Modeling Fill Blocks: As shown in Fig. 2, schemes with only diagonal blocks are not enough to provide the feasible coverage solution, for the joints of two adjacent blocks are blind areas. We heuristically employ the methods in work [6], in which “fill the gaps” blocks are distributed on two sides of the diagonal. The difference is that our solution is dynamic rather than static. Intuitively, we may model the decision of the “fill the gaps” blocks the same as the diagonal blocks. Thus, the optimization formula is presented in (16), in which the FCs serve as the binary classifiers to determine whether to fill the gap with fixed size or not. Unfortunately, this strategy will inevitably come up with a waste of resources (memristors and energy). To address this, we creatively propose a dynamic-fill scheme to prevent this limitation and further improve the utilization. In this strategy, the FCs serve as the multiclassifiers, and the output classification value stands for the fill-block size, in the form of a proportion of the current diagonal block. The illustration of two schemes (fixed-size fill and dynamic size fill) is shown in Fig. 4. Since only when the diagonal-block decision is “to start a new block,” the corresponding fill block needs to be decided, and the fill-block sequence needs to be masked by the diagonal-block sequence

\[
\begin{align*}
\arg\max_{x,z} & \quad f(p(x, z)) \\
\text{s.t.} & \quad x_i \in \{0, 1\} \\
& \quad z_i \in \{0, 1\}
\end{align*}
\]

(16)

\[
\begin{align*}
\arg\max_{x,z} & \quad f(p(x, z)) \\
\text{s.t.} & \quad x_i \in \{0, 1\} \\
& \quad z_i \in \{0, 1/m, 2/m, \ldots, 1\}.
\end{align*}
\]

(17)

Taking matrix-vector multiplication \( y = Ax \) as an example, Fig. 5 showcases the mapping of graph data on crossbar. Diagonal and fill blocks are mapped to the allowable small-scale crossbars. Based on the Kirchhoff’s current law, blocks in the same row are connected, and the corresponding subvector (splitting follows the rule of “block matrix multiplication”) is the data input of the crossbar,
The optimization objective is to maximize the expected reward of the sampled schedule scheme, and the reward function is not differentiable with respect to the controller parameters, so the policy gradient algorithm is used to construct the gradient estimates of the controller’s weight parameters. The issues that need to be addressed are as follows.

1) In such a huge action space, how to sample the block scheduling scheme directionally and heuristically.
2) Reward the sampled schedule scheme efficiently and timely, which should be instructive to the training.
3) When evaluating, multiple metrics need to be considered, including the coverage ratio, block area, which could be contradictory, as well as the trade-off between the two.

Furthermore, the reinforcement procedure is formulated as to optimize the \( \theta \) to achieve the optimal reward \( (R) \) of the candidate mapping scheme, as follows:

\[
\theta^* = \arg \max_{\theta} J(\theta) = \arg \max_{\theta} \mathbb{E}_{\pi(a_t; \theta)}(R) \tag{18}
\]

where \( J(\theta) \) is the optimization objective function, represented by the expectation of the reward achieved by coverage sequence action \( a_{1:T} \), where \( T \) is the length of action vector

\[
\nabla_{\theta} J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{\pi(a_t; \theta)} \left[ R_{
abla_{\theta}} \log \left( \pi(a_t | a_{1:t-1}; \theta) \right) \right] \tag{19}
\]

the gradient value is approximately calculated by sampling, that is,

\[
\nabla_{\theta} J(\theta) \approx \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T} \mathbb{E}_{\pi(a_t; \theta)} \left[ R_m \nabla_{\theta} \log \left( \pi(a_t | a_{1:t-1}; \theta) \right) \right] \tag{20}
\]

the algorithm of the training of the agent is presented in Algorithm 2. To clarify, in Table I, we present the notations of general reinforcement learning and their corresponding definition under current problem formulation. As for the reward \( R_m \), it involves two objectives as follows.

1) Coverage ratio relates to the performance of the computation on the crossbar.
2) Area cost relates to memristor and energy consumption. Distinctly, there exists the contradiction between the two; that is, we attempt to maximize the coverage ratio (or even complete coverage) and minimize the area cost of crossbars. To achieve this, we leverage the single-policy MORL [48], [49], [50], which resort to the scalarization function to transform the multiobjective problem into a standard single-objective one. The scalarization function \( f \) projects an objective vector \( v \) to a scalar one \( v_a = f(v, w) \) where \( w \) is a weight vector parameterizing \( f \). We employ the simple way of weight sum as the scalarization function, as follows:

\[
R_m = a \cdot \text{Coverage}(x, z) + (1 - a) \cdot \text{Area}(x, z) \tag{21}
\]

where \( a \) is the harmonic coefficient. In general, the application scenario requires complete coverage (coverage ratio is 1); consequently, what is serious is the crossbar’s consumption (area). In such a large action space, an extremely large proportion of the candidate schemes is capable to reach complete coverage, but with different area ratios. So, if we set \( a \) to be 1 or close to 1 from the beginning, the agent would tend to sample the complete coverage schemes with high probability, which even with different costs, the reward keeps consistent; thus, it makes the gradient of the controller disappear. Consequently, we must have the weighted-sum coefficient \( a \) serves as the hyperparameter that is empirically tuned. In addition, the balance of exploration and exploitation is achieved by probabilistic sampling.

C. Overall Algorithm

Our whole structure of the method is presented in Algorithm 3. The overall diagram of our modeling and optimization method is shown in Fig. 6.
Algorithm 1 Agent Sampling
Input: Null
Output: diagonal_actions, fill_actions, log_prob
1 inputs, hidden = random_initialize;
2 for i = 0 → action_len − 1 do
3 output, hidden ← LSTM(inputs, hidden);
4 logit ← the ith diagonal fcs output;
5 softmax_logits ← softmax(logit);
6 d_action ← multimodal sampling by softmax_logits; cur_log_prob ←
7 logit = log_prob + curr_log_prob;
8 diagonal_actions append d_action;
9 inputs ← output;
/*“Fill” masked by “Diagonal,” 0:
Start a new block*/;
if d_action == 0 then
10 output, hidden ← LSTM(inputs, hidden);
11 logit ← the ith fill fcs output;
12 softmax_logits ← softmax(logit);
13 d_action ← multimodal sampling by softmax_logits; cur_log_prob ←
14 logit = log_prob + curr_log_prob;
15 diagonal_actions append d_action;
16 inputs ← output;
end
end

Algorithm 2 Optimizing the Sampling Agent (REINFORCE With Baseline)
Input: log_prob, Reward
Output: Null
1 baseline ← decay * baseline + (1 − decay) * reward;
2 adv ← reward − baseline;
3 loss ← − log_prob * adv;
4 loss.backward() /*Gradients (loss is
5 differentiable w.r.t. the parameter
6 weights of the LSTM and the FCs)
7 calculated by autograd*/;
8 Gradients applied /*Gradient descend by the
9 optimizer.*/;

D. Relation and Comparisons With Other Works

In [7], a generalized reordering method to reduce the bandwidth of the matrix [8] is proposed, which can handle any asymmetrical rectangle matrix, compared with the original Cuthill–McKee reordering algorithm that can only handle the symmetrical squares. However, this work does not address the mapping scheme and its optimization. Our work does not focus on the innovation and generalization of the reordering algorithm, but rather on the automatically generating an efficient and reasonable deployment scheme after reordering a large-scale sparse matrix. In GraphR [1], the adjacency matrix of a graph is partitioned into four subgraphs with the consideration of the sparsity, and the method in GraphSAR [2] remains the same pattern that even is progressively partitioned. Their partition scheme is fixed, whereas our work makes the scheme flexible and scalable by intelligent generating. In work [6], to improve the training efficiency of sparse graph NNs on TPU device (sparse hardware), a batch of diagonal blocks and two additional batches of blocks to “fill the gap,” in which the size of the block keeps consistent, and the coverage schemes are fixed without considering the distribution of the nonzero elements. Our overall mapping framework is similar to [6] (a batch of blocks arranged along with the diagonal, and two batches fill the gaps); what is the difference is that our method is dynamic and sparsity-aware, with the consideration of the adaptability to the compilation and deployment system. Overall, compared with these related works, we adopt an intelligent generating method to achieve dynamic sparsity-aware mapping, which compares favorably with these previous works.

VI. Experiment

A. Environment

We conduct our experiments using PyTorch 1.0 framework on Intel CPU. In terms of the policy gradient training in RL, our realization relies on PyTorch’s Autograd mechanism to backwardly update the parameter weights (policy).
Fig. 7. Visualization of the datasets: QM7-5828 (22 × 22), qh882 (882 × 882), and qh1484 (1484 × 1484).

B. Dataset

We first resort to small-scale graph data, an adjacency matrix (22 × 22) that numbered 5828 in a chemical molecular dataset QM7 [51], [52]. In terms of the large-scale dataset, we experiment on two large-scale symmetric matrices, qh882 (882 × 882) and qh1484 (1484 × 1484). The visualization of these three datasets is shown in Fig. 7. In our experiments, the matrices are reordered to lower-bandwidth symmetric matrices by the Cuthill–McKee reordering algorithm as the preprocessing.

C. Metrics

\( C_{\text{ratio}} \) represents the coverage ratio of the nonzero elements of the scheme, defined as follows:

\[
C_{\text{ratio}} = \frac{\text{Nonzero_count}_{\text{mapped blocks}}}{\text{Nonzero_count}_{\text{original}}} \quad (22)
\]

it is scaled to be \([0, 1]\), and we attempt to reach 1 in our experiment, which means all nonzero elements must be mapped. \( A_{\text{ratio}} \), the area ratio of the mapping blocks to the original matrix, is defined as follows:

\[
A_{\text{ratio}} = \frac{\text{Area}_{\text{mapped blocks}}}{\text{Area}_{\text{original}}} \quad (23)
\]

it also stays in \([0, 1]\), the lower the better of the candidate schemes. Sparsity, means the utilization of the mapped crossbar of the scheme, may be viewed as the comparison metric with the original sparsity, which is defined as follows:

\[
\text{Sparsity} = \frac{\text{Nonzero_count}_{\text{mapped blocks}}}{\text{Area}_{\text{mapped blocks}}} \quad (24)
\]

Experimental results on QM7-5828 are presented in Table II, which showcases the comparison and ablation study. The comparison includes the fixed-size diagonal-block partition (Vanilla), fixed-size diagonal block with additional fill blocks (Vanilla with fill), and our proposed method “LSTM + RL” that only target the diagonal scheduling, while “LSTM + RL + Fill” additionally utilizes two series of blocks to “fill the gap.” Based on the knowledge fusion among decision points, we study the effect of BiLSTM. Experiments show that, compared with LSTM, BiLSTM achieves no significant improvements. This also happens when we significantly increase the layer number of LSTM and the hidden size. In Table III, we present the computational complexity comparison of different methods. The training of the LSTM (BPTT algorithm) costs most of the computational consumption; thus, we omit the complexity of backpropagation of FCs. Visualizations of several typical outstanding mapping schemes (complete coverage scheme with different area ratios) are shown in Fig. 8. These four results are the promising solutions (in bold) in Table II in order. The objectives (reward) optimizing curves of experiments on QM7-5828 are shown in Fig. 9. With the continuous rise of the coverage ratio, the area ratio gradually converges to a small value after the big fluctuation in the early stage. The area ratio converges to an ideal ratio, but not at the expense of the coverage ratio, which can steadily converge to 1 after 40k epochs of training under an Intel CPU (without GPU). In the later stage of our experiment, the controller is basically convergent, and the slight sampling fluctuation of the objectives is normal, because our model is a probabilistic sampling model, even if the LSTM controller parameters are fully converged, the sampling is still probabilistic.

Further experiments based on large-scale graph data/matrix (qh882 and qh1484) are shown in Table IV, in which we directly adopt “LSTM + RL + Dynamic” strategy. To reduce the scale of the problem, we partition the original matrix into grids. Considering the grid size \(k \times k\), thus, the grid
number is $N = D/k$, where $D$ is the size of the matrix. Empirically, we set the grid size of $q_{h882}$ and $q_{h1484}$ to be 32, and the fill grades number to be 4; the action space is $2^{882/32} \times 4^{882/32} = 2.4 \times 10^{24}$, and $2^{1484/32} \times 4^{1484/32} = 3.4 \times 10^{41}$, respectively, following the calculation rule of action space in Table I. As shown in Table IV, fill-block size grades number of 6 can achieve better results ($q_{h882}$: coverage ratio 1 with area ratio 0.225 and $q_{h1484}$: coverage ratio 1 with area ratio 0.171) than that of 4. That means more fine-grained fill size grades can achieve better results, and this is also expected, because the utilization of the mapped blocks is high (with lower sparsity) in this way. Visualizations of $q_{h882}$ mapping schemes are shown in Fig. 10, and the objectives optimizing curves of the promising configuration (bold in Table IV) are shown in Fig. 11. Similarly, the visualization of $q_{h1484}$ mapping scheme is shown in Fig. 12, and the corresponding objectives optimizing curves are shown in Fig. 13.

In real-world scenarios, the grid size is set subject to the allowable crossbar’s size. The complexity of the peripheral circuit also depends on the grid size and the granularity of the dynamic-fill grade number, which ultimately jointly

---

### TABLE II

**Comparison and Ablation Study Results on Small-Scale Graph QM7-5828**

| Methods         | Block size | Grid size | Fill size/grades num | Reward ratio | Schemes | Coverage ratio | Area ratio | Sparsity† |
|-----------------|------------|-----------|----------------------|--------------|---------|----------------|------------|-----------|
| Vanilla         | 4          | /         | /                    | /            | [4, 4, 4, 4, 4, 2] | 0.5          | 0.174     | 0.620     |
|                 | 6          | /         | /                    | /            | [6, 6, 6, 4]    | 0.531        | 0.256     | 0.726     |
|                 | 8          | /         | /                    | /            | [8, 8, 8, 8]    | 0.813        | 0.339     | 0.683     |
| Vanilla+Fill    | 4          | /         | /                    | /            | [4, 4, 4, 4, 4] | 0.938        | 0.445     | 0.721     |
|                 | 6          | /         | /                    | /            | [6, 6, 6, 4]    | 1.0          | 0.62      | 0.787     |
| LSTM+RL         | /          | 2         | /                    | /            | [8, 2, 12]     | 0.875        | 0.438     | 0.735     |
| LSTML+RL+Fill   | /          | 2         | /                    | /            | [6, 14]        | 0.938        | 0.337     | 0.769     |
| BiLSTM+RL+Fill  | /          | 2         | /                    | /            | [2, 2, 2, 2, 2] | 0.938        | 0.455     | 0.727     |
| LSTM+RL+Dynamic-fill | / | 2 | / | / | [2, 2, 2, 2, 2] | 0.938        | 0.455     | 0.727     |

† Sparsity of original matrix: 0.868.

### TABLE III

**Complexity Comparison of Different Methods**

| Methods† | Grid size | T | I | H | K | Complexity |
|----------|-----------|---|---|---|---|------------|
| LSTM+RL  | 2         | 12| 1 | 10| 1 | $O(T(4I+4H+2+3H+HK))$ |
| LSTM+RL +Fill | 2 | 36| 1 | 10| 1 | $O(T(4I+4H+2+3H+HK))$ |
| BiLSTM+RL +Fill | 2 | 36| 1 | 10| 1 | $O(2T(4I+4H+2+3H+HK))$ |
| LSTM+RL +Dynamic-fill | 2 | 36| 1 | 10| 1 | $O(T(4I+4H+2+3H+HK))$ |

† Taking QM7-5828 as example, T is the time step number of LSTM, I is the input size, H is the hidden size, K is the cell number.
TABLE IV
EXPERIMENTAL COMPARISON RESULTS ON LARGE-SCALE MATRIX DATASET: qH882 AND qH1484

| Dataset | Grid size | Fill grades | Reward ratio | Diagonal-blocks size | Solutions | Fill-blocks size | Coverage ratio | Area ratio | Sparsity 1 |
|---------|-----------|-------------|--------------|----------------------|----------|-----------------|----------------|-----------|------------|
| qH882   | 32        | 4           | 0.7          | 32, 32, 32, 92, 96, 64, 64, 64, 64, 96, 96, 64, 64, 18 | [2, 3, 1, 2, 2, 2, 2, 2, 3, 2] | 0.998 | 0.196 | 0.978 |
|         | 32        | 4           | 0.8          | 32, 128, 96, 128, 128, 96, 64, 64, 96, 96, 32, 50 | [2, 3, 2, 2, 2, 2, 2, 3, 2] | 0.998 | 0.204 | 0.979 |
|         | 32        | 6           | 0.7          | 32, 32, 160, 160, 128, 96, 96, 128, 32, 18 | [4, 4, 2, 3, 3, 3, 2, 5, 3] | 0.995 | 0.2 | 0.979 |
|         | 32        | 6           | 0.8          | 32, 32, 160, 96, 160, 96, 64, 82 | [2, 3, 4, 2, 3, 4, 3] | 1.0 | 0.225 | 0.955 |
| qH1484  | 32        | 4           | 0.7          | 96, 32, 32, 288, 192, 160, 64, 64, 32, 64, 32, 64, 32, 128, 32, 32, 64, 64, 32, 44 | [2, 3, 1, 2, 2, 3, 3, 2, 3, 1, 3, 3, 2, 2] | 0.992 | 0.148 | 0.981 |
|         | 32        | 4           | 0.8          | 96, 64, 64, 288, 192, 128, 96, 128, 32, 64, 32, 32, 64, 32, 128, 32, 64, 64, 12, 64, 32, 32, 64, 64, 32, 44 | [2, 2, 2, 3, 2, 3, 2, 3, 1, 3, 3, 2, 2] | 0.999 | 0.185 | 0.985 |
|         | 32        | 6           | 0.7          | 128, 224, 224, 128, 128, 160, 64, 160, 32, 128, 32, 64, 4, 0, 0, 2, 4 | [5, 2, 2, 3, 1, 3, 2, 2, 2, 4, 0, 2, 4] | 0.993 | 0.173 | 0.984 |
|         | 32        | 6           | 0.8          | 32, 96, 256, 288, 128, 96, 96, 64, 128, 160, 64, 32, 32, 64, 64, 32, 32, 64, 64, 12 | [4, 5, 2, 4, 4, 4, 5, 3, 2, 2, 3, 3, 3] | 1.0 | 0.171 | 0.984 |

1 Sparsity of qH882 is 0.995, sparsity of qH1484 is 0.997.

Unavoidably, this also increases the action space of RL, namely, the difficulty of agent optimization. These hyper-parameters can be tuned according to actual deployment scenarios, which demonstrates that our framework is flexible and scalable.

VII. Conclusion

Some recent works have made attempts at the efficient computation of sparse graphs, e.g., mapping the large-scale graph data by block coverage around the diagonal after matrix reordering. But, their mapping methods are fixed or static, instead of proposing an effective mapping schedule scheme, which should be feasible, scalable, and flexible. Based on the background of PIM crossbar computing and the characteristics of block coverage schedule problems, we put forward the principles of the mapping framework as the criterion of subsequent research. We creatively propose to formulate this problem into a sequential decision-making problem whose solution space is equal to the 0-1 integer programming problem and creatively propose the dynamic-fill method to “fill the gaps,” which well meets our proposed basic principles of the mapping framework. We model the sequential decision-making problem using a generation agent consisting of LSTM and FCs and leverage the RL algorithm (REINFORCE) to optimize it. Finally, the comparison and ablation experiments affect the area consumption of the final complete coverage solution. In principle, a smaller grid size and much more grades of the dynamic-fill blocks will result in better mapping performance (complete coverage solutions with less area cost).
on a small dataset and two large matrix data show that flexible and scalable mapping schemes can be generated with limited training epochs and time cost, and are suitable for the deployment and compilation systems. This method may also be extended to other PIM architectures, not limited to the memristive device-based platforms. Our proposed basic principles of mapping large-scale sparse graphs on memristive crossbars are worthy to be spread to the research community.

From our perspective, the limitation lies in that we only exploit the fundamental sequence-to-sequence model. Although we have tried to increase LSTM’s model scale (layers and hidden size), we did not carry out in-depth discussions and research on the knowledge fusion model, which may reach better decision-making performance. Similarly, the fundamental policy gradient algorithm is utilized. In addition, our method is based on matrix reordering, by which to reduce the matrix bandwidth. The matrix needs to be a symmetric square matrix, such as a graph adjacency matrix. Therefore, there exist some limitations in the generalization of the scheme. Currently, it is not suitable for popular image classification models, e.g., CNNs or Transformers. In future, we plan to study the fusion of the automatic mapping scheme and the sparse storage (CSC, CSR, and COO), thus further accelerating the pipeline of the graph processing on memristive crossbars of PIM/CIM platforms. Furthermore, our proposed sparsity-aware mapping learning algorithm may be further extended with the combination of sparsity [53] and fault-tolerant training, or some device-circuit nonidealities of memristive crossbars, e.g., variation and defect [54], [55], [56].

ACKNOWLEDGMENT

The statements made herein are solely the responsibility of the authors.

REFERENCES

[1] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, “GraphR: Accelerating graph processing using ReRAM,” in Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA), Feb. 2018, pp. 531–543.

[2] G. Dai, T. Huang, Y. Wang, H. Yang, and J. Wawrzynek, “GraphSAR: A sparsity-aware processing-in-memory architecture for large-scale graph processing on ReRAMs,” in Proc. 24th Asia South Pacific Design Autom. Conf., Jan. 2019, pp. 120–126.

[3] Z. Song, X. Yang, Z. Xu, and I. King, “Graph-based semi-supervised learning: A comprehensive review,” IEEE Trans. Neural Netw. Learn. Syst., early access, Mar. 18, 2022, doi: 10.1109/TNNSL.2022.3155478.

[4] W. Zhao et al., “Learning to map social network users by unified manifold alignment on hypergraph,” IEEE Trans. Neural Netw. Learn. Netw., vol. 29, no. 12, pp. 5834–5846, Dec. 2018.

[5] G. Huang, G. Dai, Y. Wang, and H. Yang, “GE-SpMM: General-purpose sparse matrix-matrix multiplication on GPUs for graph neural networks,” in Proc. SC Int. Conf. High Perform. Comput., Netw., Storage Anal., Nov. 2020, pp. 1–12.

[6] M. Bolog, B. van Merriënboer, S. Moitra, Y. Li, and D. Tarlov, “Fast training of sparse graph neural networks on dense hardware,” 2019, arXiv:1906.11796.

[7] J. Cui and Q. Qiu, “Towards memristor based accelerator for graph neural network vector multiplication,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2016, pp. 121–124.

[8] E. Cuthill and J. McKee, “Reducing the bandwidth of symmetric matrices,” in Proc. 24th Nat. Conf., 1969, pp. 157–172.

[9] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Gasco, “GPUs and the future of parallel computing,” IEEE Micro, vol. 31, no. 5, pp. 7–17, Sep./Oct. 2011.

[10] S. Ghapani, W. Ren, F. Chen, and Y. Song, “Distributed average tracking for double-integrator multi-agent systems with reduced requirement on velocity measurements,” Automatica, vol. 81, pp. 1–7, Jul. 2017.

[11] W. Ao, Y. Song, and C. Wen, “Distributed secure state estimation and control for CPSs under sensor attacks,” IEEE Trans. Cybern., vol. 50, no. 1, pp. 259–269, Jan. 2020.

[12] L. Zhang, L. He, and Y. Song, “New results on stability analysis of delayed systems derived from extended Wirtinger’s integral inequality,” Neurocomputing, vol. 283, pp. 98–106, Mar. 2018.

[13] K. Zhao, Y. Song, C. L. P. Chen, and L. Chen, “Adaptive asymptotic tracking with global performance for nonlinear systems with unknown control directions,” IEEE Trans. Autom. Control, vol. 67, no. 3, pp. 1566–1573, Mar. 2022.

[14] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, “A scalable processing-in-memory accelerator for parallel graph processing,” in Proc. 42nd Annu. Int. Symp. Comput. Archit., Jun. 2015, pp. 105–117.

[15] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, “TOP-PIM: Throughput-oriented programmable processing in memory,” in Proc. 23rd Int. Symp. High-Perform. Parallel Distribut. Comput., Jun. 2014, pp. 85–98.

[16] P. Chi et al., “Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory,” ACM SIGARCH Comput. Archit. News, vol. 44, no. 3, pp. 27–39, Jun. 2016.

[17] Y. Zhuo et al., “GraphQ: Scalable PIM-based graph processing,” in Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture, Oct. 2019, pp. 712–725.

[18] J.-J. Lin, Z. Zhu, Y. Wang, and Y. Xie, “Learning the sparsity for ReRAM: Mapping and pruning sparse neural network for ReRAM based accelerator,” in Proc. 24th Asia South Pacific Design Autom. Conf., Jan. 2019, pp. 639–644.

[19] L. O. Chua, “Memristor—the missing circuit element,” IEEE Trans. Circuit Theory, vol. CT-18, no. 5, pp. 507–519, Sep. 1971.

[20] D. B. Strukov, G. S. Snider, D. R. Stewart, and S. R. Williams, “The missing memristor found,” Nature, vol. 453, pp. 80–83, Jun. 2008.

[21] X. Hu, G. Feng, S. Duan, and L. Liu, “A memristive multilayer cellular neural network with applications to image processing,” IEEE Trans. Neural Netw. Neural Netw. Learn. Syst., vol. 28, no. 8, pp. 1889–1901, Aug. 2017.

[22] O. Krestinskaya, A. P. James, and L. O. Chua, “Neuromemristive circuits for edge computing: A review,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 1, pp. 4–23, Jan. 2020.

[23] S. Wen, R. Hu, Y. Yang, T. Huang, Z. Zeng, and Y.-D. Memristor-based echo state network with online least mean square,” IEEE Trans. Neural Netw. Learn. Syst., vol. 49, no. 9, pp. 1787–1796, Sep. 2019.

[24] C. Yakopcic, M. Z. Alom, and T. M. Taha, “Extremely parallel memristor crossbar architecture for convolutional neural network implementation,” in Proc. Int. Joint Conf. Neural Netw. (IJCNN), May 2016, pp. 1696–1703.

[25] X. Yang, B. Yan, H. Li, and Y. Chen, “ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration,” in Proc. 39th Int. Conf. Comput.-Aided Design, Nov. 2020, pp. 1–9.

[26] Z. Yan, J. Chen, R. Hu, T. Huang, Y. Chen, and S. Wen, “Training memristor-based multilayer neuromorphic networks with SGD, momentum and adaptive learning rates,” Neural Netw., vol. 128, pp. 142–149, Aug. 2020.

[27] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, “Training and operation of an integrated neuromorphic network based on metal-oxide memristors,” Nature, vol. 521, pp. 61–64, Dec. 2015.

[28] I. Kataeva, F. Merrikh-Bayat, E. Zamanidoust, and D. Strukov, “Efficient training algorithms for neural networks based on memristive crossbar circuits,” in Proc. Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2015, pp. 1–8.

[29] B. Li, Y. Wang, Y. Yang, Y. Chen, and H. Yang, “Training itself: Mixed-signal training acceleration for memristor-based neural network,” in Proc. 19th Asia South Pacific Design Autom. Conf. (ASP-DAC), Jan. 2014, pp. 361–366.

[30] M. Cheng et al., “TIME: A training-in-memory architecture for memristor-based deep neural networks,” in Proc. 54th Annu. Design Autom. Conf., Jun. 2017, pp. 1–6.

[31] S. Duan, X. Hu, Z. Dong, L. Wang, and P. Mazumder, “Memristor-based cellular nonlinear/neural network: Design, analysis, and applications,” IEEE Trans. Neural Netw. Learn. Syst., vol. 26, no. 6, pp. 1202–1213, Jun. 2015.
[32] J. Chen et al., “An efficient memristor-based circuit implementation of squeeze-and-excitation fully convolutional neural networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 4, pp. 1779–1790, Apr. 2022.

[33] A. Vaswani et al., “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5998–6008.

[34] H. Veluri, U. Chand, Y. Li, B. Tang, and A. V.-Y. Thean, “A low-power DNN accelerator enabled by a novel staircase RRAM array,” IEEE Trans. Neural Netw. Learn. Syst., early access, Oct. 20, 2021, doi: 10.1109/TNNLS.2021.3118451.

[35] B. Lyu, S. Wen, K. Shi, and T. Huang, “Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing,” IEEE Trans. Cybern., vol. 53, no. 2, pp. 1158–1169, Feb. 2023.

[36] B. Lyu, H. Yuan, L. Lu, and Y. Zhang, “Resource-constrained neural architecture search on edge devices,” IEEE Trans. Netw. Sci. Eng., vol. 9, no. 1, pp. 134–142, Jan. 2022.

[37] S. Wen et al., “Memristor-based design of sparse compact convolutional neural network,” IEEE Trans. Netw. Sci. Eng., vol. 7, no. 3, pp. 1431–1440, Jul/Aug. 2020.

[38] S. Wen et al., “CKFO: Convolution kernel first operated algorithm with applications in memristor-based convolutional neural network,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 40, no. 8, pp. 1640–1647, Aug. 2021.

[39] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2016, arXiv:1609.02907.

[40] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[41] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.

[42] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 770–778.

[43] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.

[44] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in Proc. 31st Int. Conf. Neural Inf. Process. Syst., 2017, pp. 1025–1035.

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” 2017, arXiv:1710.10903.

[46] M. Hu et al., “Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication,” in Proc. 53rd Annu. Design Autom. Conf., Jun. 2016, pp. 1–6.

[47] B. Lyu et al., “Efficient spectral graph convolutional network deployment on memristive crossbars,” IEEE Trans. Emerg. Topics Comput. Intell., vol. 7, no. 2, pp. 415–425, Apr. 2023.

[48] Z. Gábor, Z. Kalmár, and C. Szepesvári, “Multi-criteria reinforcement learning,” in Proc. 15th Int. Conf. Mach. Learn., 1998, pp. 197–205.

[49] S. Mannor and N. Shimkin, “A geometric approach to multi-criterion reinforcement learning,” J. Mach. Learn. Res., vol. 5, pp. 325–360, Dec. 2004.

[50] K. Van Mothaert, M. M. Drugan, and A. Nowe, “Scalarized multi-objective reinforcement learning: Novel design techniques,” in Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn. (ADPRL), Apr. 2013, pp. 191–199.

[51] L. C. Blum and J.-L. Reymond, “970 million druglike small molecules for virtual screening in the chemical universe database GDB-13,” J. Amer. Chem. Soc., vol. 131, no. 25, pp. 8732–8733, Jul. 2009.

[52] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, “Fast and accurate modeling of molecular atomization energies with machine learning,” Phys. Rev. Lett., vol. 108, no. 5, Jan. 2012, Art. no. 058301.

[53] C. Song, B. Liu, W. Wen, H. Li, and Y. Chen, “A quantization-aware regularized learning method in multilevel memristor-based neuromorphic computing system,” in Proc. IEEE 6th Non-Volatile Memory Syst. Appl. Symp. (NVMSA), Aug. 2017, pp. 1–6.

[54] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, “Vortex: Variation-aware training for memristor X-bar,” in Proc. 52nd Annu. Design Autom. Conf., Jun. 2015, p. 15.

[55] S. Jin, S. Pei, and Y. Wang, “A variation tolerant scheme for memristor crossbar based neural network designs via two-phase weight mapping and memristor programming,” Future Gener. Comput. Syst., vol. 106, pp. 270–276, May 2020.

[56] D. Gaol, G. L. Zhang, X. Yin, B. Li, U. Schlichtmann, and C. Zhuo, “Reliable memristor-based neuromorphic design using variation- and defect-aware training,” in Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD), Nov. 2021, pp. 1–9.