A Mapping Method Tolerating SAF and Variation for Memristor Crossbar Array Based Neural Network Inference on Edge Devices

YU MA, ShanghaiTech University, China, University of Chinese Academy of Sciences, China, and Shanghai Engineering Research Center of Energy Efficient and Custom AI IC, China
LINFENG ZHENG, ShanghaiTech University, China
PINGQIANG ZHOU, ShanghaiTech University, China and Shanghai Engineering Research Center of Energy Efficient and Custom AI IC, China

There is an increasing demand for running neural network inference on edge devices. **Memristor crossbar array (MCA)** based accelerators can be used to accelerate neural networks on edge devices. However, reliability issues in memristors, such as **stuck-at faults (SAF)** and variations, lead to weight deviation of neural networks and therefore have a severe influence on inference accuracy. In this work, we focus on the reliability issues in memristors for edge devices. We formulate the reliability problem as a 0–1 programming problem, based on the analysis of **sum weight variation (SWV)**. In order to solve the problem, we simplify the problem with an approximation - different columns have the same weights, based on our observation of the weight distribution. Then we propose an effective mapping method to solve the simplified problem. We evaluate our proposed method with two neural network applications on two datasets. The experimental results on the classification application show that our proposed method can recover 95% accuracy considering SAF defects and can increase by up to 60% accuracy with variation $\sigma = 0.4$. The results of the neural rendering application show that our proposed method can prevent render quality reduction.

CCS Concepts: • **Hardware** → **Emerging technologies**; • **Computer systems organization** → **Reliability;**

Additional Key Words and Phrases: Memristor crossbar, neural network, robustness, stuck at fault, variation

ACM Reference format:
Yu Ma, Linfeng Zheng, and Pingqiang Zhou. 2023. A Mapping Method Tolerating SAF and Variation for Memristor Crossbar Array Based Neural Network Inference on Edge Devices. *ACM J. Emerg. Technol. Comput. Syst.* 19, 2, Article 15 (May 2023), 21 pages.
https://doi.org/10.1145/3585518

1 INTRODUCTION

There has been a substantial growth of data generated by edge devices such as mobile and **Internet of Things (IoT)** devices [17]. Due to the significant data communication overhead and privacy issues, data generated from some tasks, such as real-time voice and video recognition, need to be

Authors' addresses: Y. Ma, L. Zheng, and P. Zhou, ShanghaiTech University, 393 Middle Huaxia Road, Shanghai, Shanghai, 201210, China; emails: mayu@shanghaitech.edu.cn, zhenglf@shanghaitech.edu.cn, zhoupq@shanghaitech.edu.cn.

1550-4832/2023/05-ART15 $15.00
https://doi.org/10.1145/3585518

efficiently processed on the edge side [5]. Neural networks have shown to be effective in image and voice recognition [1] in past years. As a result, there is an increasing demand of running neural network inference on edge devices.

In the inference process of neural networks, the inputs and weights are used to execute **multiply-and-accumulate (MAC)** operations, which are the most expensive operations. In order to accelerate such operations, the crossbar structure is applied which can perform MAC operations using Kirchhoff's law natively [7] by representing inputs and weights with voltages and conductances, respectively. In crossbar structures, memristor [21] can be used as a variable resistor, and has been proposed to be used in edge devices for its small on-chip area and low power dissipation [9, 22, 26]. Therefore, **memristor-based crossbar arrays (MCAs)** are suitable for accelerating neural networks on edge devices.

There exist two methods to program the conductances of memristors. One is "Close-Loop on-Device" [6] that requires expensive hardwares [14]. The other method, which is suitable for edge devices, first trains networks offline and then programs memristors to desired conductances according to the weights. This method programs the memristors with pre-calculated voltage magnitudes and pulse widths [13]. However, because of the reliability issues (i.e., defects and variations) of memristors, the programmed conductances are not equal to the desired conductances, which leads to the accuracy degradation of the neural networks [12].

The most critical reliability issue of memristors is the **stuck-at-fault (SAF)** defect [27]. If a memristor suffers from this defect, its conductance is stuck at the high resistance state (stuck-off) or low conductance state (stuck-on). In this way, the weight represented by this memristor is also fixed at the maximum or the minimum weight, respectively. The accuracy of the neural network degrades [15] because of the change in weights. In order to solve this problem, two kinds of methods are proposed. One is the retraining method [4, 15] and the other is the permutation of rows and columns of the weight matrix [23, 28]. The former re-trains networks with software, which is a computation-expensive process for computing platforms [27]. The latter studies matrix permuting methods to tolerate defects. However, both methods need SAF maps of crossbars and then perform complex algorithms to find the best permutation, which is not efficient for edge devices because of the stringent processing resources.

Another reliability issue of memristors is the variation, which originates from the deviation in the memristor's conductance when applying pre-calculated programming amplitude and width [14]. Some researchers propose to first measure variation distribution and then map the weight according to the measured distribution [4, 8, 14]. This process needs a large amount of computation which is intolerable for edge devices. As a result, *these methods are not suitable to be applied to edge devices.* Besides, these works use two memristors to represent one weight, which is a challenge for edge devices because of the limited computing resources. The quantization method [25] is also proposed to enhance the variation tolerant capability. However, the results of this work show that this method can only tolerate a small variation [20, 29].

All previous works don't consider the limitation of the crossbar size. Although the crossbar size can be large in ideal, it is always limited in reality [2] because of the power and reliability issues [19]. Therefore, multiple crossbars are integrated using a shared interconnect, such as **Networks on Chip (NoCs)** (see Figure 1). Large weight matrices are always separated into pieces and then mapped to several MCAs [3]. Then the peripheral circuit combines the results of the MCAs. In this paper, we propose a mapping method to tolerate both SAF and variation of memristors while considering both the crossbar size limitation and the stringent resources of edge devices. The contributions of this paper are summarized as follows.

- We analyze MCA-based edge devices with two features. One is to apply one memristor representing one weight method according to the characteristic of edge devices – limited

resources. The other is to apply vector-vector multiplication, which allows us to flexibly assign weights to different MCAs to perform MAC operations. According to the aforementioned circumstances, we analyze the weight deviation caused by SAF and variation using the **sum weight variation (SWV)** metric. We formulate the SWV optimization problem as a 0–1 programming problem.

- Since the 0–1 programming is hard for edge devices to get a suitable solution, we use a local mapping method to reduce the SWV and study its influence on neural networks. As the local mapping method can only get limited improvement, we simplify the 0–1 programming problem based on an observation – different columns have similar values in neural networks. Then we propose an optimal remapping method for the simplified problem and modify the computing pipeline of MCAs.
- We evaluate our proposed method with two neural network applications – classification and neural rendering. The experimental results on the MNIST dataset show that the proposed method can increase the reliability of MCAs for edge devices, the inference accuracy can be maintained even if the SAF rate reaches 28%, and our proposed method can tolerate variation with $\sigma = 0.4$. The experimental results on the FERN dataset show that our proposed method can obviously reduce the reliability influence on the rendered image quality.

The rest of this paper is organized as follows. In Section 2, we first introduce MCA-based neural network inference according to characteristics of edge devices and then introduce the models of the SAF defect and variation. Then we present the problem formulation in Section 3. After that, we introduce our proposed method based on the characteristics of neural networks for edge devices in Section 4. Finally, we show our experimental results in Section 5, followed by the conclusion in Section 6.

2 BACKGROUND

Edge devices, which are used to execute real-time tasks such as translation and objective recognition, have stringent computing resources and a tight power budget [5]. MCA-based accelerators can be used in edge devices. In this section, we first introduce the circuitry structure of the crossbar for edge devices. Then we demonstrate the neural network computing and weight programming methods. Lastly we introduce the SAF defect and variation models of memristors which lead to errors in computing and neural network inference.

2.1 MCA Architecture

When running neural network inference on edge devices, MCA-based computing architecture is considered as candidate hardware for accelerating neural networks [9] (see Figure 1). Each MCA performs MAC operations and the peripheral circuits perform activation, routing, and other operations.

2.2 Memristor Crossbar Based Computing

Tasks that are too large for one MCA are first split into pieces and the split tasks are mapped to several MCAs. Then the peripheral circuit combines the results generated by the MCAs and outputs the final result. The acceleration process for a given neural network is shown as follows.

- Firstly, the neural network is trained offline and Figure 2(a) shows a three-layer network and the weight matrix of the first layer as an example. In this work, the row and column numbers of the weight matrix are the input and output sizes of the layer, respectively.

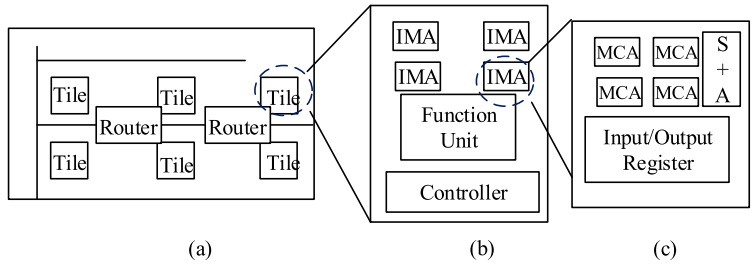


Fig. 1. The architecture of MCA based devices. Tile and IMA (in-situ multiply-accumulate) are higher hierarchies of MCAs [18].

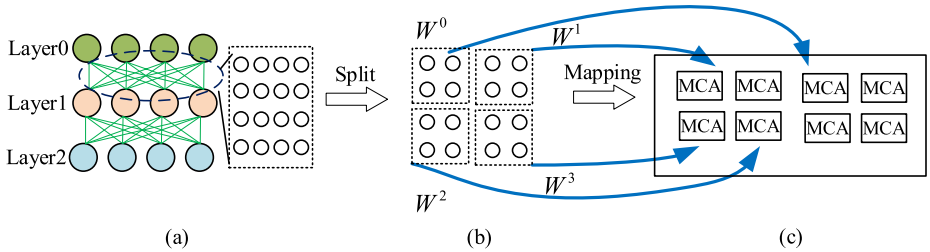


Fig. 2. The architecture of MCA based devices. (a) neural network, (b) split network, (c) map weight to different MCAs.

- Before executing the inference process, the weights of neural networks need to be mapped to MCAs. Because of the limited size of MCAs [24], the weight matrix is firstly split into pieces (see Figure 2(b)).
- Then each piece of the weight matrix is mapped to an MCA (see Figure 2(c)). In this process, the conductances of memristors are programmed according to weights. *Although one weight can be represented by multiple memristors to increase the robustness of computing [27], because of the limited resources in edge devices, the number of memristors should be as few as possible.* We consider the case that one memristor represents one weight (see Figure 3(a)) in our work. Each gate line (see Figure 3(b)) controls transistors of a certain column. In the programming phase, each memristor is programmed to a certain conductance according to the weight. In the inference phase, transistors of gate lines are open, and current can flow through memristors.

Suppose we represent one weight with one memristor, the conductance of the memristor is programmed linearly according to weights (see Figure 3(b)) which can be expressed as Equation (1) [30]:

$$G = \frac{G_{\max} - G_{\min}}{W_{\max} - W_{\min}}(W - W_{\min}) + G_{\min} \tag{1}$$

where $G_{\min}$ and $G_{\max}$ are the minimum and maximum conductances of the memristor, respectively, $W_{\min}$ and $W_{\max}$ are the minimum and maximum weights, respectively.

After mapping weights to MCAs, in the inference phase, the inputs are first fed to **input registers (IRs)** of MCAs. For a certain layer, in each cycle, each bit of the input is fed to the crossbars and every column in the crossbars executes MAC operations in parallel according to Kirchhoff's law, $I = \vec{G} \cdot \vec{V}$ (see Figure 3(a)). The result of every cycle is shifted and added in shift-and-add unite (S+A) (see Figure 1(c)). After all the bits are computed, the result of each MCA is combined to get

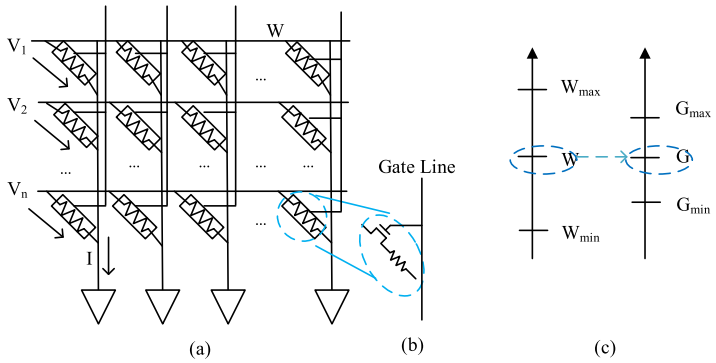


Fig. 3. The MAC operation with memristor crossbar array. (a) Schematic of the 1T1R array and MCA operation. (b) Schematic of the 1T1R cell. (c) Linear one-to-one mapping.

the result of the layer. After activation, the result is transferred to the next layer. After the last layer is processed, the inference result is generated.

2.3 Defect and Variation Modeling

Ideally, the conductance of a memristor can be programmed according to a given weight (see Equation (1)). However, there are defects and variations in memristors. In this work, we consider two reliability issues of memristors - SAF defect and variation.

2.3.1 Stuck at Fault. SAF defect is the most critical reliability issue in memristor crossbar [27]. Assume that we want to program the conductance of one memristor to G, if the memristor suffers from the SAF defect, the conductance state of the memristor cannot be changed and is stuck at high (stuck-on) or low (stuck-off). Considering the SAF defect, the programmed conductance can be expressed as Equation (2):

$$G' = \begin{cases} G_{\text{min}}, \text{ stuck-off} \\ G, \text{ no defect} \\ G_{\text{max}}, \text{ stuck-on} \end{cases} \tag{2}$$

where G' is the conductance we program in practice. If the memristor suffers from this issue, the weight which is programmed to this memristor is fixed because the conductance represents the weight. In this case, the accuracy of the given neural network degrades [15].

2.3.2 Variation. In addition to SAF defect, variation also has a huge influence on the inference accuracy of MCA-based neural networks [4, 14]. As the conductance is programmed with pre-calculated voltage magnitude and pulse width, the conductance we program in practice follows a lognormal distribution [11] which can be expressed as Equation (3):

$$G' = G \cdot e^{\theta} \tag{3}$$

where G' is the programmed conductance in practice, G is the conductance we want to program, and $\theta \sim N(0, \sigma^2)$ is the variation distribution.

2.3.3 SWV Metric. Because of the SAF defect and variation of memristors, the effective weights represented by the memristors don't equal the weight that we want to program. In order to evaluate the programming error, we use the SWV [14] metric to calculate the difference between weight

(W) and effective weight (W^{eff}):

$$SWV = \sum_i \left| w_i^{eff} - w_i \right| \tag{4}$$

where W_i is the i-th weight value of the weight matrix. The larger SWV is, the larger the error and the accuracy loss are.

3 PROBLEM ANALYSIS

We first formulate the minimization of SWV caused by SAF and variation issues as an optimization problem. In this section, we optimize the SWV metric by assigning weights to different MCAs in the mapping process. Then we propose our method to solve this problem in the next two sections.

3.1 SWV Analysis

At the system level, digital results from MCAs are gathered and processed before producing the output of one layer because the weight matrices are split into pieces. As a result, we consider the piece of weight matrix which is mapped to the k-th MCA, i.e.:

$$G = \frac{G_{\max} - G_{\min}}{W_{\max}^k - W_{\min}^k} \left(W - W_{\min}^k \right) + G_{\min} \tag{5}$$

where $W_{\min}^k$ and $W_{\max}^k$ are the minimum and maximum weights in the k-th MCA. As there are SAF defects and variations of memristors in MCAs, the weights which are programmed to memristors are not accurate. We analyze the weight deviation in the programming phase. If we program one memristor to conductance G, we can represent the conductance deviation as ΔG ($= G' - G$) because of the SAF defect and variation. According to Equation (5), the effective weight W^{eff} can be expressed as:

$$W^{eff} = W_{\min}^k + \frac{W_{\max}^k - W_{\min}^k}{G_{\max} - G_{\min}} (G + \Delta G - G_{\min}) \tag{6}$$

As a result, considering one certain weight, the error between the changed weight and the initial weight can be expressed as:

$$\left| W^{eff} - W \right| = \frac{W_{\max}^k - W_{\min}^k}{G_{\max} - G_{\min}} \cdot |\Delta G| \tag{7}$$

Then, considering that the weight matrix is split into several IMAs, the SWV of one weight matrix is the sum of SWVs of every IMA which can be expressed as:

$$\begin{aligned} SWV &= \sum_{k \in MCAs} SWV_k \\ &= \sum_{k \in MCAs} \sum_{i \in MCA_k} \frac{W_{\max}^k - W_{\min}^k}{G_{\max} - G_{\min}} \cdot |\Delta G_{k,i}| \end{aligned} \tag{8}$$

where $\Delta G_{k,i}$ is the conductance deviation at the i-th position in the k-th MCA. In Equation (8), $G_{\max}$ and $G_{\min}$ are constants and ΔG is subject to a distribution because of the SAF defect and variation. Therefore, we can minimize the sum of the $(W_{max} - W_{min})$ of each MCA to minimize SWV:

$$\min SWV \sim \min \sum_k \left(W_{\max}^k - W_{\min}^k \right) \tag{9}$$

As a result, our objective is to minimize the range of the weights which are mapped to each MCA while ensuring that MAC operations can be executed. For convenience, we denote our optimization objective as sum(max-min).

3.2 Problem Formulation

In this section, we develop a mathematical model to describe the problem. As aforementioned, we can reduce the difference between the maximum and minimum values of every crossbar to reduce the reliability influence. The problem is described from Equations (10) to (18). In order to perform matrix-vector multiplication, we should ensure that the mapping method meets the following three requirements.

- Every weight is mapped and only mapped once.
- The items in the same row are mapped to the same row of the MCA.
- The items in the same column are mapped to the same column of the MCA.

$$\min_{\delta_{i,j,k}} \quad \Sigma_k \left(max_{i,j}(\delta_{i,j,k} \cdot w_{i,j}) - min_{i,j}(\delta_{i,j,k} \cdot w_{i,j}) \right) \tag{10}$$

$$\text{s.t.} \quad \delta_{i,j,k} \in \{0,1\}, \quad \forall i,j,k \tag{11}$$

$$\sum_k \delta_{i,j,k} = 1, \quad \forall i,j \tag{12}$$

$$l_{k,j} \in \{0,1\} \quad \forall k,j \tag{13}$$

$$\sum_j l_{k,j} = S, \quad \forall k \tag{14}$$

$$\sum_i \delta_{i,j,k} = l_{k,j} \cdot S, \quad \forall j,k \tag{15}$$

$$r_{k,i} \in \{0,1\} \quad \forall k,i \tag{16}$$

$$\sum_i r_{k,i} = S, \quad \forall k \tag{17}$$

$$\sum_j \delta_{i,j,k} = r_{k,i} \cdot S, \quad \forall i,k \tag{18}$$

where $i = 0, 1, \ldots, R$, $j = 0, 1, \ldots, C$, $k = 0, 1, \ldots, K$, R and C are numbers of rows and columns of one weight matrix, respectively, S is the size of MCA (assume MCA is square) and $K = \lceil R/S \rceil \times \lceil C/S \rceil$ is the number of MCAs to use, $w_{i,j}$ is the i-th row j-th column weight in the weight matrix, $l_{k,j} = 1$ indicates that there are weights in j-th column mapped to the k-th MCA, $r_{k,i} = 1$ indicates that there are weights in i-th row mapped to the k-th MCA, and $\delta_{i,j,k}$ is to record whether $w_{i,j}$ is mapped to the k-th MCA.

Taking Figure 4 as an example, the weight matrix (size 4×4) is split into four pieces (the size of each piece is 2×2): top left, top right, bottom left, and bottom right. Each piece is mapped to one MCA. The marks (k) of the MCAs are 0, 1, 2, 3, respectively. For example, in MCA1, there are four memristors representing four weights. The bottom left memristor represents the weight in the second row, the third column. As a result, $\delta_{1,2,1} = 1$ and $\delta_{1,2,k}(k \neq 1) = 0$. Besides, only elements in the second and third columns are mapped to this MCA. As a result, $l_{1,2}$ and $l_{1,3}$ are both 1 and $l_{1,k}(k \notin 2, 3) = 0$. The r symbol is similar to l, for example, $r_{1,2} = 1$ and $r_{1,3} = 0$.

Equation (12) ensures that every weight is mapped and can be only mapped to one MCA, Equation (14) ensures that S columns are mapped to each MCA and Equation (15) ensures that S items in j-th column are mapped to k-th MCA. Equation (17) ensures that S rows are mapped to each MCA and Equation (18) ensures that S items in i-th row are mapped to k-th MCA. The Equations (14), (15), (17), and (18) ensure that we can execute matrix-vector multiplication in MCAs, i.e., the weights, that are mapped to the same column in one MCA, are in the same column of weight matrix.

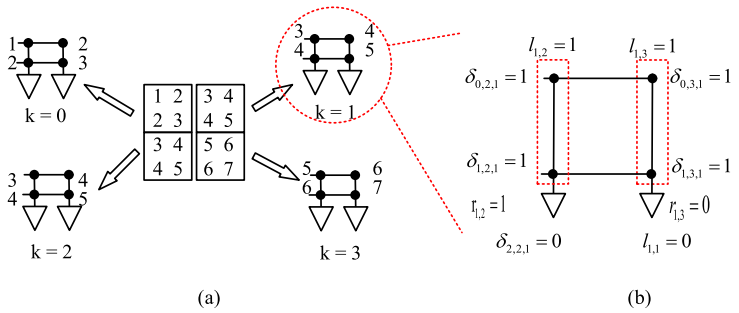


Fig. 4. 0–1 programming problem analysis. The crosspoint is the abstraction of the memristor device. (a) One mapping solution. (b) An example of parameters in the problem formulation.

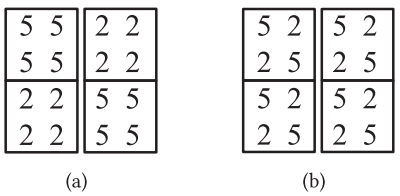


Fig. 5. Two weight map examples: (a) describes a special case; (b) describes a general case.

3.3 Local Mapping Method

The 0–1 programming problem described in Section 3.2 is hard to solve, especially on edge devices. The most direct method is to use the local maximum and minimum weight values in weight mapping, rather than use the global weight. In this way, the $W^k_{\max}$ can be reduced while the $W^k_{\min}$ can be increased. As a result, the optimization objective can be reduced. For example, Figure 5(a) shows an example that the weights with similar values are gathered. If we use the global mapping method, the objective function is $(5-2) \times 4$. This value can be reduced to 0 if we use the local mapping method. In this case, we can simply use the local maximum and minimum values to reduce the loss.

However, in real cases, the weight distribution is quite even. For example, if the weight distribution is Figure 5(b), the local mapping method fails to reduce the loss. Thus, we need to further modify the distribution of the weights to make the mapping method robust.

4 COLUMN BY COLUMN METHOD

Generally, computing with all columns in parallel leads to high power and more computing resources, i.e., large fan-out of DACs and high frequency of ADCs. If all columns are computed together, each layer can be computed in 10^3ns [18], which is overpowered for edge devices. Secondly, the power consumption is too large for an edge device. Furthermore, sneak path [31] is also an import reliability issue. *Considering these three aspects, we apply column by column computing method in our work (this satisfies the performance requirement which will be introduced in Section 4.3).* We further provide a new direction to improve the reliability of MCA-based neural network inference for edge devices.

We have formulated the weight mapping problem for matrix-vector multiplication in the previous section. In this section, we reduce the constraints by using vector-vector multiplication and propose our method for solving this problem. As a result, the mapping method should meet the following two requirements.

- Every weight is mapped and only mapped once: Equation (21).
- The items in the save column are mapped to the same column of the MCA: Equations (22)–(24).

We can formulate this problem as an 0–1 programming problem as Equations (19)–(24).

$$\min_{\delta_{i,j,k}} \quad \Sigma_k(max_{i,j}(\delta_{i,j,k} \cdot w_{i,j}) - min_{i,j}(\delta_{i,j,k} \cdot w_{i,j})) \tag{19}$$

$$\text{s.t.} \quad \delta_{i,j,k} \in \{0,1\}, \quad \forall i,j,k \tag{20}$$

$$\sum_k \delta_{i,j,k} = 1, \quad \forall i,j \tag{21}$$

$$l_{k,j} \in \{0,1\} \quad \forall k,j \tag{22}$$

$$\sum_j l_{k,j} = S, \quad \forall k \tag{23}$$

$$\sum_i \delta_{i,j,k} = l_{k,j} \cdot S, \quad \forall j,k \tag{24}$$

The new 0–1 programming problem is still a difficult optimization problem and inappropriate for edge devices to solve. As a result, we propose a simple approximate solution to this problem.

4.1 Problem Simplification

In order to simplify the problem, we analyze the weight distribution of neural networks. We analyze one layer (size 784×1000) of a well-trained network as an example. We sort weights in each column and then compare the difference of weights among the sorted columns. Then we calculate the **normalized average difference (NAD)** between two columns, i.e., average difference divided by the range of the two columns. We find that the NAD of the most different two columns is less than 10%. As for the two most similar columns, the NAD is smaller than 0.5%, the distributions of which are shown in Figure 6. To summarize, in neural networks, we observe that weight distributions in different columns are very similar. In order to simplify the programming problem (19), *we approximately assume that different columns have the same weights.* For example, as shown in Figure 7(a), weights in every column are $\{1, 2, 3, 4, 5\}$.

4.2 Mapping Method

Our purpose is to map similar values to the same MCA with the assumption that different columns have the same weights. Our proposed mapping method is shown in Algorithm 1. Given a weight matrix with size $R \times C$ and MCA size S, our proposed method is to determine which weight should be mapped to which MCA. Firstly, we compute the number of MCAs that we need to map the weight as $\lceil R/S \rceil \times \lceil C/S \rceil$ (line 1) and judge whether the weight matrix can fill all the MCAs assigned for it (line 2). If not, we add rows or columns to the end of the weight matrix to fill all MCAs (line 3). Then we sort the values of the weight matrix by columns and store the sorting index matrix (line 6). At last, we map the sorted matrix to MCAs in order (line 8) and return the mapping label δ.

For example, Figure 7(a) shows a 5×5 matrix we need to map and the size of MCAs is 3. Initially, the weight matrix is chaotic (see Figure 7(a)). Then we need $\lceil 5/3 \rceil \times \lceil 5/3 \rceil = 4$ MCAs and then insert one dummy column and one dummy row for this matrix (see Figure 7(b)). Thus, MCA0 - MCA3 are assigned for this weight matrix. After that, we sort the weights by columns (see Figure 7(c)). At last, we assign the weight matrix to MCAs as shown in Figure 7(d). Then we can get the mapping label δ. For example, the location $(0, 0)$ is mapped to MCA0 and location $(4, 4)$ is mapped to MCA3, therefore $\delta_{0,0,0} = 1$ and $\delta_{4,4,3} = 1$.

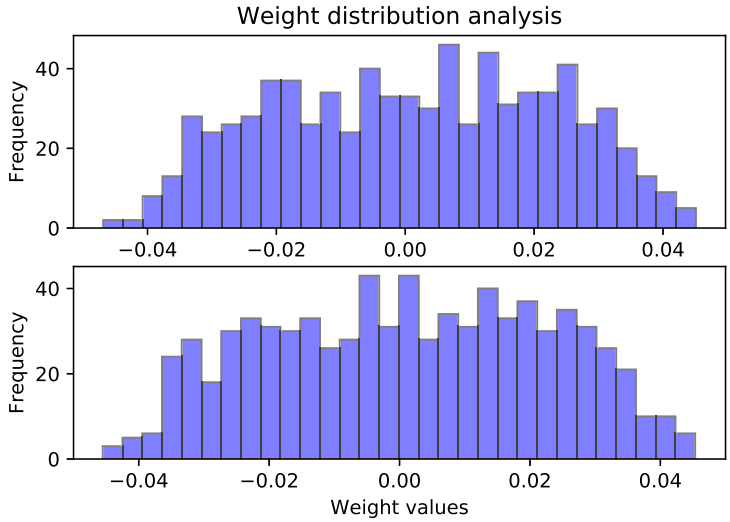


Fig. 6. Weight distribution of the two most similar columns.

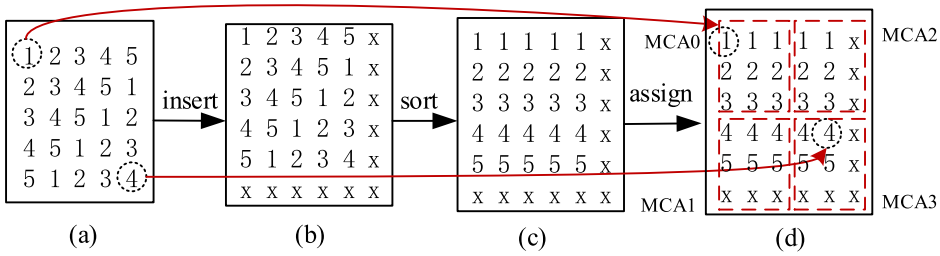


Fig. 7. Mapping method.

ALGORITHM 1: Mapping Method.

Input: Weight matrix W (with R rows and C columns), MCA crossbar size S
Output: δ

```
Compute number of needed MCAs;
if R or C is not divisible by S then
    Insert maximum values;
end if
for all Columns do
    Sort and store index matrix I;
end for
Assign weights to MCAs in sequence;
Return δ
```

In our solution, all the weights are mapped only once. Thus, Equation (21) is satisfied. We don't exchange weights between two columns. As a result, Equations (23) and (24) are also satisfied. Besides, our proposed method only needs a sorting operation which is very simple and therefore suitable for edge devices. Our proposed method is optimal for the simplified problem (because of

the limited space, we don't prove it here). Thus, our proposed method can increase the robustness of the computing considering variation and SAF.

Proof. Assume that the sizes of MCAs are S and we have K MCAs. We first consider one column case, i.e., we have $K \times S$ weights in one column. Assume that $A_1, A_2, A_3, \ldots, A_{KS}$ are sorted weights in this column and $1, 2, 3, \ldots, KS$ are indexes of the weights. For a mapping solution, we denote as $[S_i](i = 1, 2, 3, \ldots, K)$ where S_i is the set of weight that mapped to the i-th MCA. Let l_i and r_i be the index of the minimum and maximum weights which are mapped to the i-th MCA. We have:

$$SWV = \sum_{i=1}^{k} (A_{r_i} - A_{l_i}) = \sum_{r=1}^{k} \sum_{j=l_i}^{r_i - 1} (A_{j+1} - A_j)$$

Defining a set $T = \bigcup_{i=1}^{K} \{r | l_i \leq r \leq r_i - 1, r \in \mathbb{Z}\}$, we have:

$$SWV = \sum_{j \in T} A_{j+1} - A_j$$

Define $P = \{p | 1 \leq p \leq KS, p \neq kS, k = 1, 2, 3, \ldots, K\}$. For any $p \in P$, assume that $p = kS + \alpha, \alpha = 1, 2, 3, \ldots, S - 1$, there exists a set T whose $l_i \leq p$ and $r_i > p$. If there isn't, $\forall l_i \leq p, r_i \geq p$. In other words, the first p weights is divided by several sets, which is in conflict with $p = km + \alpha$. As a result, $\forall p, \exists i, p \in [l_i, r_i - 1]$. Thus, $P \subseteq T$. Thus, we have:

$$SWV = \sum_{j \in T} (A_{j+1} - A_j) \geq \sum_{j \in P} (A_{j+1} - A_j)$$

The set P is our solution. As a result, our proposed method is optimal if we only have one column. According to our assumption that different columns have the same weights. We sort all columns and then map to MCAs that don't increase the cost. As a result, our proposed method is optimal for the simplified problem. □

4.3 Computing Pipeline

As we compute with MCA column by column, we need to modify the computing pipeline from the conventional method which is introduced in Section 2. Input data are stored in the input registers (IRs). The required data are sent to IRs by routers cycle by cycle. In the first cycle, data are fed to IRs by routers. In the following cycles, the first column is computed by the MCA bit by bit. After that, the second column is computed, and so on. For example, Figure 8 shows the input of one layer and the weight assignment. Figure 8(a) shows the input and Figure 8(b) shows which data are fed to which MCA. The upper data are fed to MCA0 and MCA2, the lower data are fed to MCA1 and MCA3 according to our mapping label (generated in the mapping phase) by routers in the first cycle. Assuming that the input data are 16-bit, in the 2^{nd} to 17^{th} cycle, the first columns of layers of the MCAs are computed bit by bit. In the next 16 cycles, the second columns are computed. After all the columns are computed, peripheral circuits gather the results of MCAs and transfer the activation to the next layer.

5 EXPERIMENTAL RESULTS AND DISCUSSION

In order to evaluate the effectiveness of our proposed method, we study two applications on two datasets. On each dataset, we compare the neural network inference accuracy with and without our proposed method considering SAF and variation. On the other hand, our proposed method introduces overhead to the accelerator. As a result, we discuss the overhead of our proposed method after our experimental results.

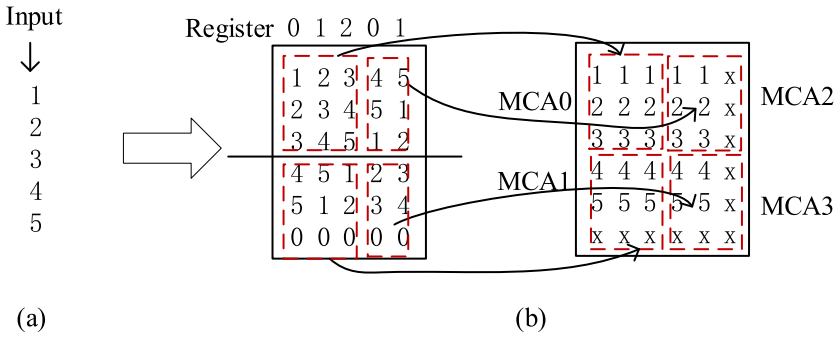


Fig. 8. Computing pipeline.

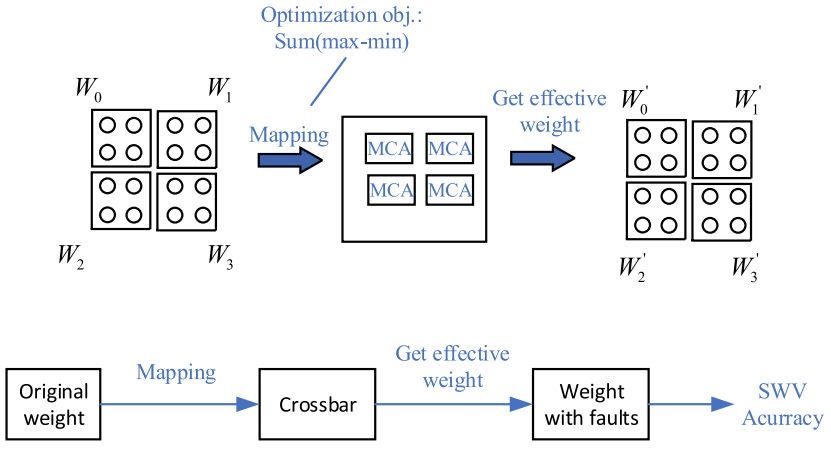


Fig. 9. Experiment flow.

In our experiment, the minimum and maximum nominal resistance of memristors are set to $10k$ and $1M$ Ω, respectively. The SAF defect and variation are modeled with Equation (2) and (3) respectively. We assume that the SAF defects are distributed randomly in each MCA and 81.6% of the defects are stuck-on and 18.4% of the defects are stuck-off [15]. The size of MCAs is set to 64 [24] in the classification task and 32 in the neural rendering task. We use the Monte-Carlo simulation method in our experiment and evaluate three cases.

- NO: use the conventional mapping method.
- LC: use the local mapping method.
- OP: use our proposed remapping method.

As simulating the memristor crossbar is time-consuming, we use the effective weight in our experiment. Our experiment flow is shown in Figure 9. Firstly, we map the weight matrix to the crossbar and then compute the effective weight by inputting vectors. For example, we can get the first row of the effective weight by inputting $[1, 0, 0, \ldots]$ to our in-house simulator. Then we use the effective weight to test the accuracy of the MNIST dataset and the rendering quality of the FERN dataset. We have verified that the results from this way match the results from the simulator. As a result, we use our simulator to simulate the effective weight rather than simulating the whole computing process for ease of evaluation.

Table 1. Properties of Neural Networks on MNIST

Number of layers	Weight Matrix Dimension	Ideal Accuracy
2	784×10	91.47%
3	784×256, 256×10	95.73%

Table 2. The Property of Each Layer in NeRF

#Weight matrix	1	2	3	4	5	6	7	8	9	10	Output
Size	60	256	256	256	316	256	256	256	280	128	3+1

Table 3. Sum(max-min) Results of the Three Layer Network

Method	NO	LC	OP
layer1	17.49	10.13	2.01
layer2	4.77	4.17	1.31

We evaluate our proposed method in the following three aspects.

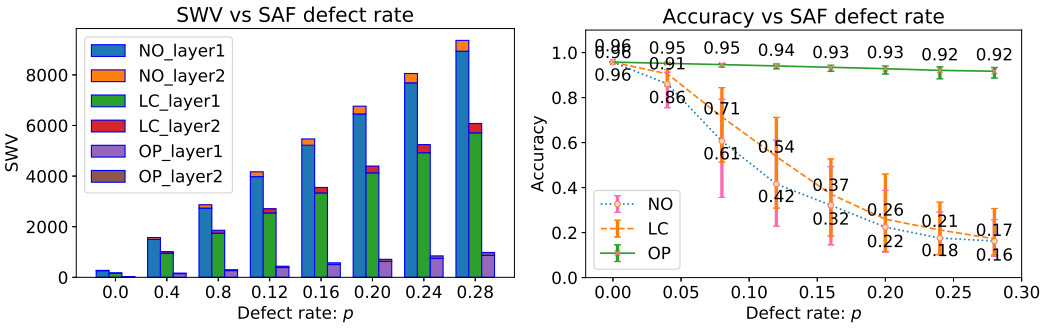
- *Performance – three-layer network on the MNIST dataset.* We evaluate the effectiveness of our proposed method with different SAF defect rates with the three-layer network.
- *Scalability – two-layer network on the MNIST dataset.* As our proposed method is sensitive to weight matrix and MCA size, we analyze the scalability of our proposed method with the two-layer network.
- *Usability – eleven-layer network on the FERN dataset.* To show that our proposed method works in larger datasets and other applications, we evaluate our proposed method on the FERN dataset in the neural rendering task [16].

We first quantitatively evaluate our proposed method with two feed-forward neural networks (see Table 1) on the MNIST dataset [10] following the practice of previous works [4, 27]. To prove the usability of our proposed method, we qualitatively evaluate our proposed method with an eleven-layer network on the FERN dataset to perform the rendering task. The input size of each layer and the output size of the last layer are shown in Table 2.

5.1 Performance

5.1.1 Results for SAF Defect. We evaluate the effectiveness of our method for the SAF defect with varying defect rate p from 4% to 28%. Firstly, we evaluate the optimization objective in our method, sum(max-min), and Table 3 shows the result. The local mapping method can reduce about 42% and 12.5% sum(max-min) for layer one and layer two, respectively. Our proposed remapping method can further reduce sum(max-min) about 5× times and 3 times for the two layers. The SWV reduction is similar to the sum(max-min) reduction as shown in Figure 10(a).

The accuracy results with different defect rates are shown in Figure 10(b). Without our proposed method, the accuracy of the network is influenced greatly and degrades below 80% with only a 5% defect rate. If the defect rate increases to about 20%, the accuracy is about 20%. The local mapping method has little accuracy improvement compared with the conventional mapping method. With our proposed remapping method, the inference accuracy is almost not influenced (maintains about 95% accuracy) with 5% defect rate and can be maintained at high accuracy (92%) with even 28% defect rate.



(a) SWV vs SAF defect rate with the 3-layer network. (b) Accuracy VS. SAF defect rate with the 3-layer network.

Fig. 10. Results for the SAF defect: (a) describes the relationship between the SWV and the defect rate; (b) describes the relationship between the accuracy and the defect rate.

Table 4. Comparison with other Methods using 3-layer Neural Network and 20% Defect Memristors

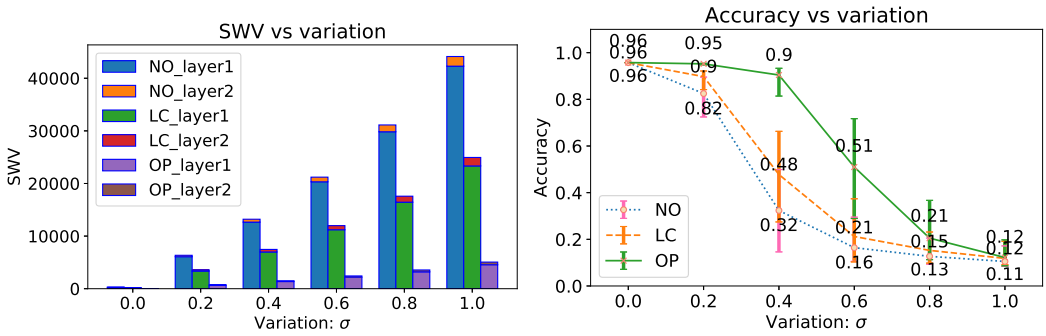
Method	[27]	[15]	Ours
Recovered accuracy	30.1%	95%	95.9%

Besides, as shown in Table 4, we compare our proposed method with state of the art methods [15, 27] with 20% defect rate. The recovered accuracy is defined as accuracy normalized with the ideal classification accuracy of the neural network [27]. With stringent resources (one memristor represents one weight), our proposed method reaches 95.9% recovery accuracy. [27] can only get 30.1% while this method can get 97.6% recovered accuracy with multiple memristors representing one weight. Although [15] can also reach 95% recovery accuracy, our proposed method doesn't need the retraining process which is a very costly process for platforms [27]. As a result, our method is friendly for edge devices and can reach state-of-the-art effectiveness.

5.1.2 Results for Variation. In order to evaluate the effectiveness of our proposed method with different variations (see Equation (3)), we test σ from 0.2 to 1.0 with the three-layer network. The SWV and accuracy results are shown in Figure 11. The SWV result of variation of local mapping method and remapping method is similar to the result for SAF.

As for the accuracy, with the normal mapping method, the accuracy degrades lower than 85% with $\sigma = 0.2$. The local mapping method can maintain a high accuracy of 90% with $\sigma = 0.2$. However, the improvement is not obvious. Our proposed remapping method can maintain a very high accuracy (95%) with $\sigma = 0.2$, and a high accuracy (90%) with $\sigma = 0.4$. Our proposed method represents one weight with one memristor while previous works [4, 14] use two memristors to represent one weight. As a result, we can't compare our proposed method with them. The effectiveness of our proposed method is limited with high σ in this experiment because the layer 256×10 has a small number of rows.

Quantization methods [20, 29] can also be applied to mitigate the variation influence. However, the quantization method can only be applied to small variation applications. On MNIST dataset and with a 3-layer fully-connected network, the state-of-the-art method [29] can only increase the accuracy from 50% (without the method) to 80% (with the method). When the accuracy reduces to about 30%, the quantization method can hardly increase the accuracy. Meanwhile, our method can



(a) SWV vs σ with 3-layer network.

(b) Accuracy vs σ with 3-layer network.

Fig. 11. Results for the variation: (a) describes the relationship between the SWV and the variation; (b) describes the relationship between the accuracy and the variation.

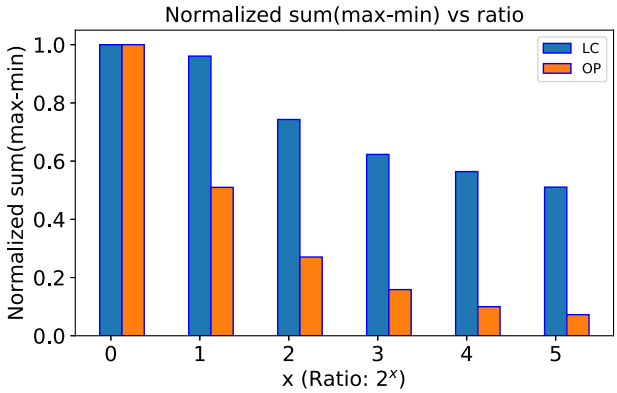


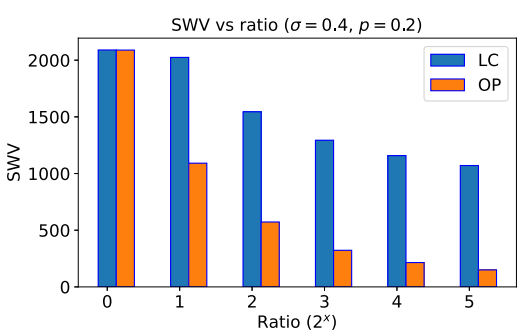
Fig. 12. sum(max-min) vs σ with 2-layer network.

increase the accuracy from 32% to 90% (see Figure 11(b)). As a result, our proposed method has better performance than quantization methods. Besides, our proposed method doesn't increase the difficulty of the training process and can mitigate both SAF and variation problems.

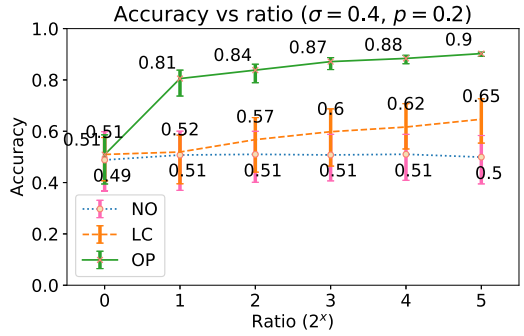
5.2 Scalability

The effectiveness of our method is influenced by the size of the crossbar (S) and the number of rows of the weight matrix (R). In this section, we study how the ratio (R/S) influences the effectiveness of our proposed method with the two-layer network (see Table 1). We test the effectiveness of our method with different ratios by varying the crossbar size. In this experiment, we use two reliability problem cases: 1) 20% SAF defect rate with variation $\sigma = 0.4$, 2) 20% SAF defect rate with variation $\sigma = 0.8$. We show the results in Figures 12 and 13.

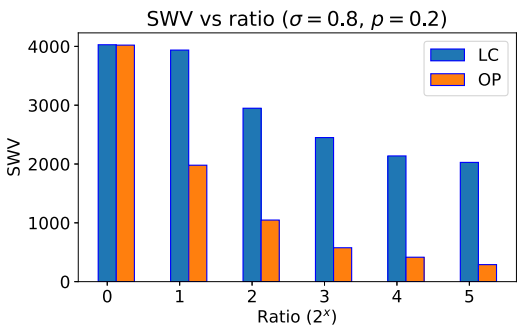
Similar to the evaluation for SAF and variation, we first study our normalized optimization objective as shown in Figure 12. When $ratio = 2^0 = 1$, the weight matrix is mapped to one crossbar. As a result, the local mapping method and the remapping method don't improve the results. As the ratio increase, the local mapping method improves the results slightly. Even if the ratio increases to $2^5 = 32$, the local mapping method can only reduce about 50% loss. Our proposed remapping method can reduce about 90% loss with $ratio = 32$. This result also proves our observation – the



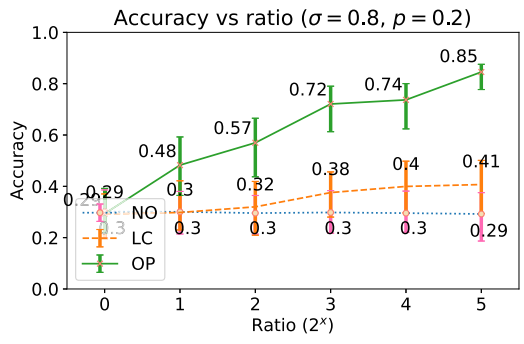
(a) SWV vs ratio: $P = 0.2$, $\sigma = 0.4$.



(b) Accuracy vs ratio: $P = 0.2$, $\sigma = 0.4$.



(c) SWV vs ratio: $P = 0.2$, $\sigma = 0.8$.



(d) Accuracy vs ratio: $P = 0.2$, $\sigma = 0.8$.

Fig. 13. SWV and accuracy vs ratio under 20% SAF defect rate with small $\sigma = 0.4$ and 0.8, respectively. (a) and (c) describe the relationship between the SWV and the ratio; (b) and (d) describe the relationship between the accuracy and the ratio.

weight value is distributed evenly in the weight matrix. The method without remapping can hardly reduce the reliability influence.

As shown in Figure 13, under large variation $\sigma = 0.8$ with SAF defect, the accuracy with our proposed method increases as the ratio increases. Under small variation ($\sigma = 0.4$) with SAF defect, our proposed method can get a high accuracy (81.88%) when $ratio = 2$, and the accuracy also increases when the ratio increases.

Although our proposed method doesn't work with small ratios, in practice, the size of neural networks is quite large and thus the ratio is also large. As a result, our proposed method is supposed to work well in practice. In the next section, we study the usability of our method on a larger dataset in another neural network application.

5.3 Usability

The effectiveness of our proposed method is based on the weight distribution pattern shown in Section 4.2. The pattern is similar in the same task (regardless of the dataset). As a result, in order to evaluate the usability our proposed method, we study another complicated neural network application with our proposed method. Neural rendering is one of the most studied topics using neural networks recently. Meanwhile, the rendering task needs to infer the sample points' properties of the image. And one image has tens of millions of data (sample points). In other words, the

Table 5. Sum(max-min) Results for NeRF

	1	2	3	4	5	6	7	8	9	10
NO	18.81	81.87	118.2	120.44	163.16	132.39	151.66	37.95	145.89	8.16
LC	14.42	59.85	78.84	83.46	104.5	80.44	101.04	29.07	65.21	6.18
OP	8.41	13.6	17.93	18.32	21.97	18.69	24.08	4.71	15.4	2.12

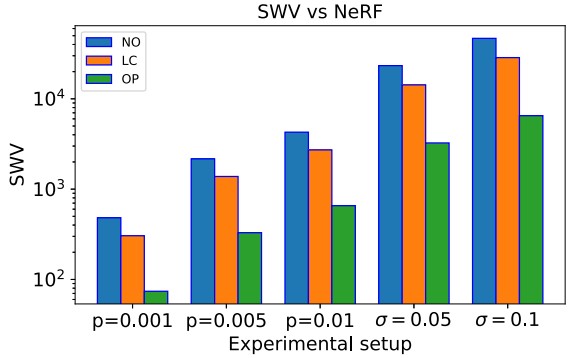


Fig. 14. The SWV of different experimental setups in NeRF.

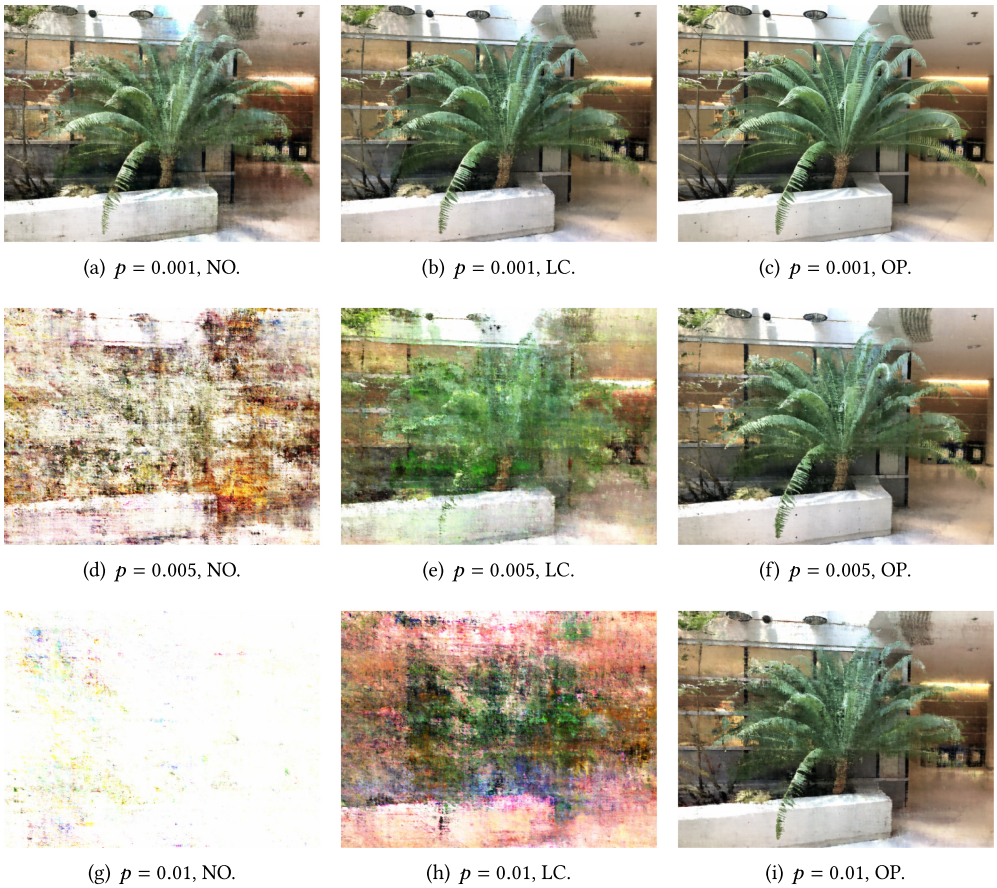
rendering of one image is to individually execute the inference process for tens of millions of data. Meanwhile, the large-scale classification dataset ImageNet has about 15 million data, which is the same scale as the data quantity of one image in the rendering task. As a result, we use the neural rendering task to evaluate the usability of our proposed method. We train a neural network for the rendering purpose, the input size of which is shown in Table 2. As the rendering network is deep and the application is noise-sensitive, we use small defect rates and small variation σ in our experiment when evaluating our proposed method in the neural rendering application.

Table 5 shows the results of our optimization objective. The local mapping method can reduce the objective by 36% compared with the conventional method. Our proposed remapping method can further reduce the loss by about 77%.

The SWV results are shown in Figure 14. The local mapping method can reduce by about 70% SWV for every layer. As the network computation error is the product of errors of all layers, we can reduce the network computation result exponentially. As a result, our proposed method can improve the rendered image quality, which is shown in Figures 15 and 16. With the conventional mapping method, we can only recognize the rendered images with $p = 0.001$. The local mapping method increases little rendering quality with all the experimental settings. Our proposed remapping method hardly reduces the rendering quality with $p = 0.001$ and has obvious improvement with all the experimental setups.

5.4 Discussion

5.4.1 Speed. According to the computing pipeline in Section 4.3, we analyze the computing speed with our proposed method. Without loss of generality, every cycle is set to 100ns [18], and therefore, each layer of neural networks can be computed within 10^6ns if the size of MCA is 100. Because of the pipeline design, *this system can reach* 10^3 ***frames per second (FPS)****. Generally, the computing power for real-time applications is about* 10^2 *FPS. Although our computing method causes a decrement in the processing speed, our method can still work in real-time mode for edge devices.*



(a) $p = 0.001$, NO. (b) $p = 0.001$, LC. (c) $p = 0.001$, OP.

(d) $p = 0.005$, NO. (e) $p = 0.005$, LC. (f) $p = 0.005$, OP.

(g) $p = 0.01$, NO. (h) $p = 0.01$, LC. (i) $p = 0.01$, OP.

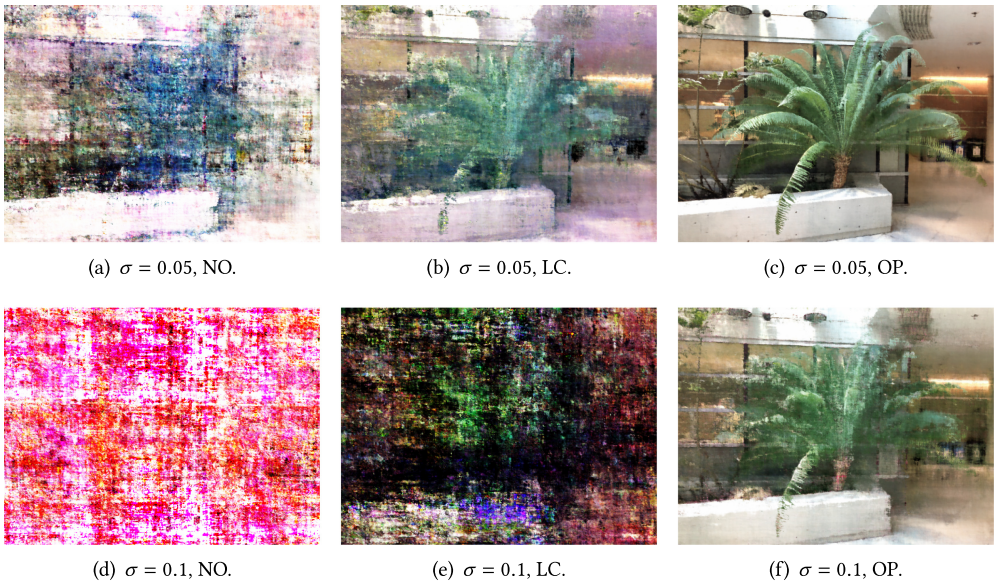
Fig. 15. A set of SAF results for NeRF: (a) shows NO method with defect rate 0.001; (b) shows LC method with defect rate 0.001; (c) shows OP method with defect rate 0.001; (d) shows NO method with defect rate 0.005; (e) shows LC method with defect rate 0.005; (f) shows OP method with defect rate 0.005; (g) shows NO method with defect rate 0.01; (h) shows LC method with defect rate 0.01; (i) shows OP method with defect rate 0.01.

5.4.2 Energy. Our proposed method needs to re-arrange inputs, which may lead to workload overhead for the peripheral circuits (such as controller, routers, and DACs) of the circuit. Considering that our method doesn't introduce extra computation operations of crossbars, the energy overhead should be low.

5.4.3 Storage. Our proposed method needs an extra area to store the mapping solution in the system. Considering that tasks which are executed with edge devices should not be too complicated, the networks should also not be complicated. As a result, the extra storage is small. Taking one weight matrix with 1024×1024 as an example, the device only needs $1024 \times 1024 \times log(1024)bit < 2MB$ storage.

6 CONCLUSION AND FUTURE WORK

In this paper, we have discussed two reliability issues of MCAs for edge devices. Considering the limitation of MCA size, we have formulated the reliability problem as a 0–1 programming problem



(a) $\sigma = 0.05$, NO. (b) $\sigma = 0.05$, LC. (c) $\sigma = 0.05$, OP.

(d) $\sigma = 0.1$, NO. (e) $\sigma = 0.1$, LC. (f) $\sigma = 0.1$, OP.

Fig. 16. A set of variation results for NeRF: (a) shows NO method with $\sigma = 0.05$; () shows LC method with $\sigma = 0.05$; (c) shows OP method with $\sigma = 0.05$; (d) shows NO method with $\sigma = 0.1$; (e) shows LC method with $\sigma = 0.1$; (f) shows OP method with $\sigma = 0.1$.

based on the analysis of SWV for edge devices. Based on the observation – different columns have similar weights, we have simplified the problem and proposed an optimal mapping method for the simplified problem. The experimental results on two datasets have shown that our proposed method can prevent the accuracy loss for neural networks. In our future work, we will optimize the whole computing system to execute our proposed method more effectively. Besides, our proposed method can work in tandem with other techniques such as quantization methods. We will combine our proposed method with them in our future work.

REFERENCES

[1] Mahbubul Alam, Manar D. Samad, Lasitha S. Vidyaratne, Alexander M. Glandon, and Khan M. Iftekharuddin. 2020. Survey on deep neural networks in speech and vision systems. *Neurocomputing* 417 (2020), 302–321. https://doi.org/10.1016/j.neucom.2020.07.053

[2] Amirali Amirsoleimani, Fabien Alibart, Victor Yon, Jianxiong Xu, M. Reza Pazhouhandeh, Serge Ecoffey, Yann Beilliard, Roman Genov, and Dominique Drouin. 2020. In-memory vector-matrix multiplication in monolithic complementary metal–oxide–semiconductor-memristor integrated circuits: Design choices, challenges, and perspectives. *Advanced Intelligent Systems* 2, 11 (2020), 1–23. https://doi.org/10.1002/aisy.202000115

[3] Indranil Chakraborty, Mustafa Fayez Ali, Dong Eun Kim, Aayush Ankit, and Kaushik Roy. 2020. GENIEx: A generalized approach to emulating non-ideality in memristive xbars using neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218688

[4] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. 2017. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 19–24. https://doi.org/10.23919/DATE.2017.7926952

[5] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. 2020. A survey of accelerator architectures for deep neural networks. *Engineering* 6, 3 (2020), 264–274. https://doi.org/10.1016/j.eng.2020.01.007

[6] Miao Hu, Hai Li, Yiran Chen, Qing Wu, and Garrett S. Rose. 2013. BSB training scheme implementation on memristor-based circuit. In *2013 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. 80–87. https://doi.org/10.1109/CISDA.2013.6595431

[7] 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. 2016. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2897937.2898010

[8] Song Jin, Songwei Pei, and Yu Wang. 2020. A variation tolerant scheme for memristor crossbar based neural network designs via two-phase weight mapping and memristor programming. *Future Generation Computer Systems* 106 (2020), 270–276. https://doi.org/10.1016/j.future.2020.01.021

[9] Olga Krestinskaya, Alex Pappachen James, and Leon Ong Chua. 2020. Neuromemristive circuits for edge computing: A review. *IEEE Transactions on Neural Networks and Learning Systems* 31, 1 (2020), 4–23. https://doi.org/10.1109/TNNLS.2019.2899262

[10] Yann LeCun, Corinna Cortes, and C. J. Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]* 2 (2010). http://yann.lecun.com/exdb/mnist.

[11] Seung Ryul Lee, Young-Bae Kim, Man Chang, Kyung Min Kim, Chang Bum Lee, Ji Hyun Hur, Gyeong-Su Park, Dongsoo Lee, Myoung-Jae Lee, Chang Jung Kim, U-In Chung, In-Kyeong Yoo, and Kinam Kim. 2012. Multi-level switching of triple-layered TaOx RRAM with excellent reliability for storage class memory. In *2012 Symposium on VLSI Technology (VLSIT)*. 71–72. https://doi.org/10.1109/VLSIT.2012.6242466

[12] Bing Li, Bonan Yan, Chenchen Liu, and Hai (Helen) Li. 2019. Build reliable and efficient neuromorphic design with memristor technology. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASPDAC'19)*. Association for Computing Machinery, New York, NY, USA, 224–229. https://doi.org/10.1145/3287624.3288744

[13] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Tingwen Huang, Qing Wu, and Mark Barnell. 2014. Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 63–70. https://doi.org/10.1109/ICCAD.2014.7001330

[14] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. 2015. Vortex: Variation-aware training for memristor X-bar. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2744769.2744930

[15] Chenchen Liu, Miao Hu, John Paul Strachan, and Hai (Helen) Li. 2017. Rescuing memristor-based neuromorphic design with high defects. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062310

[16] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (Dec. 2021), 99–106. https://doi.org/10.1145/3503250

[17] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. 2018. Edge computing for the internet of things: A case study. *IEEE Internet of Things Journal* 5, 2 (2018), 1275–1284. https://doi.org/10.1109/JIOT.2018.2805263

[18] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 14–26. https://doi.org/10.1109/ISCA.2016.12

[19] Manjunath Shevgoor, Naveen Muralimanohar, Rajeev Balasubramonian, and Yoocharn Jeon. 2015. Improving memristor memory with sneak current sharing. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*. 549–556. https://doi.org/10.1109/ICCD.2015.7357164

[20] Chang Song, Beiye Liu, Wei Wen, Hai Li, and Yiran Chen. 2017. A quantization-aware regularized learning method in multilevel memristor-based neuromorphic computing system. In *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. 1–6. https://doi.org/10.1109/NVMSA.2017.8064465

[21] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. 2008. The missing memristor found. *Nature* 453, 7191 (2008), p.80–83. https://doi.org/10.1038/nature06932

[22] Shikhar Tuli and Shreshth Tuli. 2020. AVAC: A machine learning based adaptive RRAM variability-aware controller for edge devices. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS45731.2020.9180670

[23] Lixue Xia, Wenqin Huangfu, Tianqi Tang, Xiling Yin, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, and Huazhong Yang. 2018. Stuck-at fault tolerance in RRAM computing systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8, 1 (2018), 102–115. https://doi.org/10.1109/JETCAS.2017.2776980

[24] Qi Xu, Song Chen, Hao Geng, Bo Yuan, Bei Yu, Feng Wu, and Zhengfeng Huang. 2020. Fault tolerance in memristive crossbar-based neuromorphic computing systems. *Integration* 70 (2020), 70–79. https://doi.org/10.1016/j.vlsi.2019.09.008

[25] Qing Yang, Hai Li, and Qing Wu. 2018. A quantized training method to enhance accuracy of ReRAM-based neuromorphic systems. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS.2018.8351327

[26] Tommaso Zanotti, Francesco Maria Puglisi, and Paolo Pavan. 2020. Reconfigurable smart in-memory computing platform supporting logic and binarized neural networks for low-power edge devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2020), 1–1. https://doi.org/10.1109/JETCAS.2020.3030542

[27] Baogang Zhang, Necati Uysal, Deliang Fan, and Rickard Ewetz. 2019. Handling stuck-at-faults in memristor crossbar arrays using matrix transformations. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASPDAC'19)*. Association for Computing Machinery, 438–443. https://doi.org/10.1145/3287624.3287707

[28] Baogang Zhang, Necati Uysal, Deliang Fan, and Rickard Ewetz. 2020. Handling stuck-at-fault defects using matrix transformation for robust inference of DNNs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2020), 2448–2460. https://doi.org/10.1109/TCAD.2019.2944582

[29] Chenguang Zhang and Pingqiang Zhou. 2021. A quantized training framework for robust and accurate ReRAM-based neural network accelerators. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC'21)*. New York, NY, USA, 43–48. https://doi.org/10.1145/3394885.3431528

[30] Shuhang Zhang, Grace Li Zhang, Bing Li, Hai Helen Li, and Ulf Schlichtmann. 2019. Aging-aware lifetime enhancement for memristor-based neuromorphic computing. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1751–1756. https://doi.org/10.23919/DATE.2019.8714954

[31] Mohammed Affan Zidan, Hossam Aly Hassan Fahmy, Muhammad Mustafa Hussain, and Khaled Nabil Salama. 2013. Memristor-based memory: The sneak paths problem and solutions. *Microelectronics Journal* 44, 2 (2013), 176–183. https://doi.org/10.1016/j.mejo.2012.10.001

Received 26 January 2022; revised 1 December 2022; accepted 29 January 2023