Reliability-Driven Neuromorphic Computing Systems Design

Qi Xu[1], Junpeng Wang[1], Hao Geng[2], Song Chen[1], Xiaoqing Wen[3]
[1]School of Microelectronics, University of Science and Technology of China
[2]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[3]Department of Computer Science and Networks, Kyushu Institute of Technology
{xuqi@,wjp97@mail,songch@}ustc.edu.cn, hgeng@cse.cuhk.edu.hk, wen@cse.kyutech.ac.jp

***Abstract*—In recent years, memristive crossbar-based neuromorphic computing systems (NCS) have provided a promising solution to the acceleration of neural networks. However, stuck-at faults (SAFs) in the memristor devices significantly degrade the computing accuracy of NCS. Besides, memristors suffer from process variations, causing the deviation of actual programming resistance from its target resistance. In this paper, we propose a novel reliability-driven design framework for a memristive crossbar-based NCS in combination with general and chip-specific design optimizations. First, we design a general reliability-aware training scheme to enhance the robustness of NCS to SAFs and device variations; a dropout-inspired approach is developed to alleviate the impact of SAFs; a new weighted error function, including cross-entropy error (CEE), the l_2-norm of weights, and the sum of squares of first-order derivatives of CEE with respect to weights, is proposed to obtain a smooth error curve, where the effects of variations are suppressed. Second, given the neural network model generated by the reliability-aware training scheme, we exploit chip-specific mapping and re-training to further reduce the computation accuracy loss incurred by SAFs. Experimental results clearly demonstrate that the proposed method can boost the computation accuracy of NCS and improve the NCS robustness.**

I. Introduction

Neuromorphic computing systems (NCS) based on hardware designs intend to mimic neuro-biological architectures. Different from conventional von Neumann architectures, NCS has often been constructed with highly parallel, extensively connected, and collocated computing and storage units. Thus the gap between CPU computing capacity and memory bandwidth is eliminated [1]. However, the implementation of NCS with the CMOS technology suffers from a severe mismatch between NCS building blocks (neurons and synapses) and CMOS primitives (Boolean logic gates). Recently, the emerging memristive technology is adopted to implement the synapse circuit thanks to the similarity between memristive and synaptic behaviors [2]. For example, a memristor is suitable to store the weight of a synapse because its resistance can be programmed by applying current or voltage. Furthermore, compared with state-of-the-art CMOS designs, memristive crossbars have been proven as one of the most efficient nanostructures that carry out matrix-vector multiplications while hardware cost and computation energy are significantly reduced [3].

Despite of these significant advantages, NCS implementations on memristive crossbars are faced with some reliability challenges. Firstly, memristors suffer from stuck-at faults (SAFs), which make them stuck at high or low resistance states, resulting in a significant yield loss in NCS. Secondly, in a memristor-based NCS, programming the resistance of a memristors induces stochastic device variations [4]. As a result, the actual programmed resistance deviates from its target resistance and finally results in significant errors to the output of the neural network.

To tolerate SAFs, a number of solutions have been proposed. Huangfu *et al*. [5] proposed a mapping algorithm for tolerating SAFs by using extra memristive crossbars. Xu *et al*. [6] introduced a mapping-based heuristic algorithm to derive a weight-memristor mapping for fault tolerance. Based on a row flipping transformation, a neuron permutation transformation, and a value range transformation, a matrix transformation framework [7] was developed by Zhang *et al*. to handle SAFs. But redundant memristors are needed in these mapping-based fault tolerant methods, thus inevitably causing high hardware overhead. Recently, machine learning-based techniques have been successfully utilized in addressing the SAF problem. Liu *et al*. [8] presented a re-training based method to tolerate SAFs. Xia *et al*. [9] proposed a fault tolerant training with a re-mapping technique by using the sparsity of neural networks. However, these solutions focus only on SFAs but overlook programming variations.

In addition, to address stochastic device variations, Liu *et al*. [10] developed a variation tolerant training technique by adjusting the training goal according to the impact of variations. Thus, a set of pre-trained neural networks are generated. By testing the network models, the best one is applied to the memristive crossbar. However, for a crossbar with stochastic variations, the optimal network model can hardly be derived. Chen *et al*. [11] investigated a fault and variation tolerant framework for memristive crossbar-based NCS. It first finds an optimal mapping between weights and memristors for SAFs and variation tolerance. Then, a re-training process based on the conventional gradient descent technique is further adopted to tune weights. However, the resistance variation values of memristors are known in advance, thus the method is essentially not different from the SAF-aware approaches. What's worse, when the variation model changes, inference accuracy of the network decreases. This means the above variation tolerant methods lack robustness against variations.

Although recent works have investigated SAFs and variations tolerance, they only focus on the chip-specific design of a memristive crossbar-based NCS. In other words, these SAF tolerant solutions can be derived only if the exact locations of SAFs in a crossbar are known in advance. Although the SAF locations in a crossbar can be obtained by March-C or squeeze-search based testing methods [12], [13], the test overhead is too high for an NCS implemented with many memristive crossbars. Therefore, a general reliability design of a memristive crossbar-based NCS without prior testing is required. Fundamentally different from previous approaches, we propose a novel reliability-driven design framework for a memristive crossbar-based NCS in combination with general and chip-specific design optimization. To the best of our knowledge, this is the first NCS reliability-aware training flow capable of addressing both SAFs and variations simultaneously. Key technical contributions of this work are listed as follows.

- We propose a general reliability-aware training scheme to enhance the robustness of an NCS to SAFs and device variations. A dropout-inspired approach is developed to alleviate the impact of SAFs. A new weighted error function, including the cross-entropy error (CEE), the l_2-norm of weights, and the sum of squares of first-order derivatives of CEE with respect to weights, is designed to obtain a smooth error curve, where the effects of device variations are suppressed.
- Given the neural network model generated by the reliability-aware training scheme, we further propose a chip-specific design to restore the computation accuracy loss incurred by SAFs. We analyze the sensitivity of weights to SAFs in the specified neural network. Based on the sensitivity measurement and SAF locations

- in a crossbar, a weight-memristor mapping based on neuron permutation is proposed to prevent highly sensitive synapses from being mapped to faulty memristors.
- A Monte Carlo simulation is exploited to evaluate the performance of the proposed framework on different datasets. Experimental results show that our method not only improves the robustness of an NCS against SAFs and device variations, but also boosts the computation accuracy of an NCS.

The remainder of this paper is organized as follows. Section II presents the preliminary and introduces the problem to be addressed in this paper. Section III introduces our overall design flow. Section IV and Section V describe the proposed reliability-driven design framework for neuromorphic computing systems. Section VI presents experimental results, followed by conclusions in Section VII.

II. Preliminaries

A. Fault and Variation Models in Memristors

Due to the immature fabrication technology, manufacturing reliability is still a major concern for a memristive crossbar-based NCS. Memristor faults can be divided into soft faults and hard faults. For a soft fault, the resistance of an RRAM cell is not correct but can still be tuned. On the contrary, for a hard fault, the resistance of a memristor cannot be changed. Thus, the computational accuracy of an NCS is limited. Among all kinds of hard faults, stuck-at-one (SA1) faults and stuck-at-zero (SA0) faults appear rather frequently. Permanent open switch defects and broken word-lines lead to SA1 faults, which put memristors into a high resistance state. Meanwhile, SA0 faults, which are caused by over-forming defects, reset failures, or short defects, force on-chip memristors into a low resistance state [13]. Based on published data [11], 83.8% of SAFs are SA1 faults and 16.2% of SAFs are SA0 faults.

Various types of variations, which are divided into two categories, i.e., parametric variation and switching variation, are observed in memristors. Device-to-device parametric variation is caused by fabrication imperfection, such as random discrete dopants and line edge roughness [14]. On the other hand, switching variation is a cycle-to-cycle variation triggered by driving circuit design. Any small fluctuations in the magnitude and pulse-width of the programming current or voltage can cause resistance variations. Throughout this paper, we use matrix $\boldsymbol{C}$ to represent the resistance states of a crossbar. The work in [15] have reported that the programmed resistance states of memristors follow a lognormal distribution, which is shown in Equation (1):

$$\tilde{c}_{ij} = c_{ij} \cdot \exp(\theta_{ij}), \ \ \theta_{ij} \sim \mathcal{N}(0, \sigma^2), \tag{1}$$

where $\tilde{c}_{ij}$ and c_{ij} are the actual resistance and the target resistance of a memristor in the i-th row and j-th column of $\boldsymbol{C}$, respectively, θ_{ij} denotes the zero-mean Gaussian variation with a variance of σ. Therefore, the memristor resistance variation is a multiplicative deviation. An example of a memristive crossbar with SAFs and variations is shown in Fig. 1.

B. Problem Formulation

In this work, we employ the computation robustness and accuracy to quantify the performance of an NCS.

Definition 1 (Computation Robustness)**.** *A robust NCS means that any small disturbance of network weights will not change the output of the neural network.*

Definition 2 (Computation Accuracy)**.** *The computation accuracy of an NCS is the probability that the trained NCS successfully classifies the test samples.*

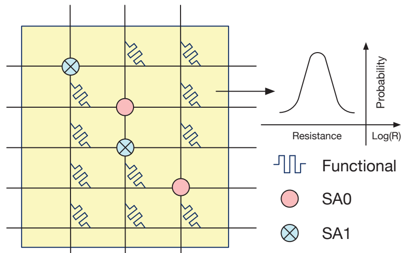


Fig. 1 A memristive crossbar with SAFs and device variations.

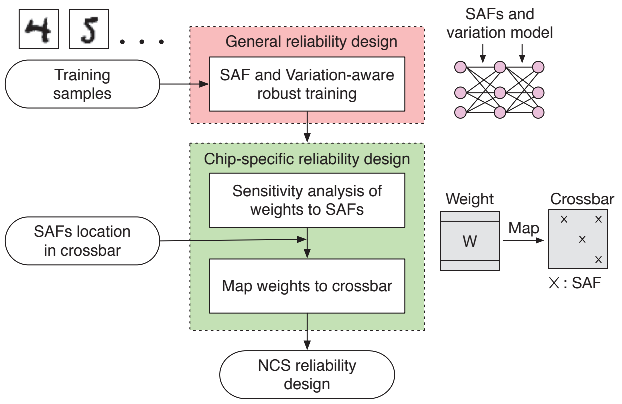

Training samples
General reliability design
SAF and Variation-aware robust training
SAFs and variation model
Chip-specific reliability design
Sensitivity analysis of weights to SAFs
SAFs location in crossbar
Map weights to crossbar
Weight
W
Map
Crossbar
X : SAF
NCS reliability design

Fig. 2 The proposed reliability-driven design framework for an NCS.

Based on the above metrics, we define the problem of the fault and variation tolerance in an NCS (FVTN) as follows.

Problem 1 (FVTN)**.** *Given a series of datasets and a device variation model, we implement an NCS on a memristive crossbar with compensation for the impact of faults and device variations, so that the computation robustness and the computation accuracy of the NCS are improved.*

III. Overall Flow

As shown in Fig. 2, the overall flow of the proposed reliability-driven design framework for an NCS consists of two stages: (1) general design optimization and (2) chip-specific design optimization. In the general design optimization stage, we propose a general reliability-aware training scheme for an NCS. Through a dropout-inspired technique and a new weighted error function, the robustness of an NCS against SAFs and device variations is enhanced. In the second stage, a chip-specific design is further proposed to reducce the computation accuracy loss incurred by SAFs. Based on the neural network model generated by the reliability-aware training, we analyze the sensitivity of each weight with respect to SAFs. After the locations of SAFs in the crossbar are pinpointed by the testing method, a weight-memristor mapping is performed to prevent the highly sensitive synapses from being mapped to the SAFs. Next, a network re-training algorithm is performed to compensate for the computation accuracy loss, while maintaining the robustness against variations. As a result, a reliability design of a memristive crossbar-based NCS is achieved.

IV. Reliability-Driven Network Training

In this section, we first introduce our general reliability-aware training algorithm. After that, we analyze the loss behaviors of the neural network.

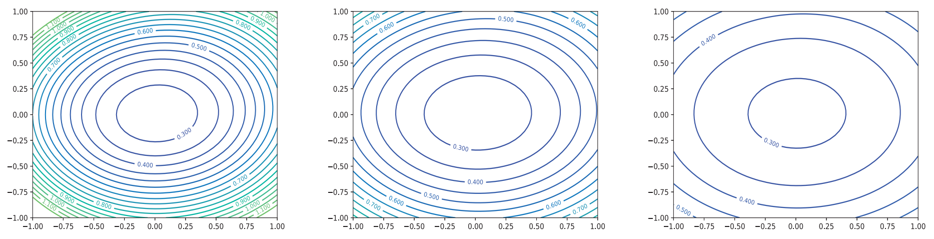


Fig. 3 2D visualization of the loss surface of a two-layer MLP trained with different regularizers on MNIST dataset: (a) only the cross-entropy error; (b) with the l_2-norm regularizer; (c) the proposed training method.

A. General Reliability-aware Network Training

In the general reliability-aware training scheme, a dropout-inspired technique and a new weighted error function are developed to learn more robust features about SAFs and variations. Note that the reliability-aware training is an off-device method, which means the neural network is trained at software level and the trained model is directly deployed on memristive crossbars with SAFs and device variations.

To alleviate the complicated dependencies among weights and make a network robust against SAFs, a dropout-inspired technique is proposed. We train the network with the mini-batch gradient descent (MGD) approach, which can provide a more efficient computation process than stochastic gradient descent (SGD). A group of instances are randomly picked from the training set in each training iteration. For each batch, we sample a network with random SAFs distribution. Following previous works [11], [16], the weight with an SA0 (SA1) fault is temporarily set to the maximum (minimum) value in the current network. Note that the weights with SAFs will not participate in the back-propagation. Hence, the operation of injecting SAFs into weights is similar to the dropout strategy in deep learning [17], where the neurons are temporarily removed from the network. As a result, a weight in the network does not rely on other specific weights, and the complex co-adaptation of weights is avoided.

In addition, a new weighted error function is designed to obtain a smooth error curve, where the effects of variations are suppressed. Generally, regularization is adopted to control the complexity of the network model and avoid over-fitting. The regularization term can be expressed by adding a penalty to the error function of the learning algorithm as shown in Equation (2):

$$\tilde{E}(\boldsymbol{W}) = E(\boldsymbol{W}) + \lambda\Omega, \tag{2}$$

where $\boldsymbol{W}$ is the weight matrix with the size of $m \times n$, and the parameter $\lambda > 0$ controls the relative importance of the data-dependent error $E(\boldsymbol{W})$ (typically cross-entropy error in classification problems) and regularization penalty Ω.

Two popular regularizers are the l_1-norm regularizer and the l_2-norm regularizer as shown in Equation (3):

$$\Omega_{l_1}(\boldsymbol{W}) = \|\boldsymbol{W}\|_1, \Omega_{l_2}(\boldsymbol{W}) = \|\boldsymbol{W}\|_2^2. \tag{3}$$

In order to avoid confusion, all norms $\|\cdot\|$ are calculated with respect to flattened vectors. The l_1-norm regularizer has the property that some weights w_{ij} are driven to zeros, leading to a sparse network model. Compared with the l_1-norm regularizer, the l_2-norm regularizer gives more bias to the large weights during training and shrinks the weight distribution to the small value range [18]. To compensate for the impact of faults and device variations, a smooth error curve is needed. According to the work [19], the l_2-norm can reduce the sensitivity of the network and thus enhance the training robustness.

In addition, if the first-order derivatives of the error function with respect to weights are adopted as a regularizer, it disfavors error functions that change rapidly. Therefore, the first-order derivative regularizer helps to avoid sharp changes in error (and hence in output) with minor changes in weights. Furthermore, since the memristor resistance variation is a multiplicative deviation, the weight perturbations are proportional to weight magnitudes. Hence we should consider the weight value in the first-order derivative term. In accordance with the argument, we derive our objective function by combining Equation (2), the l_2-norm regularizer in Equation (3), and the first-order derivative term as follows:

$$\min_{\boldsymbol{W}} \ E(\boldsymbol{W}) + \lambda_1 \|\boldsymbol{W}\|_2^2 + \lambda_2 \left\| \boldsymbol{W} \odot \frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{W}} \right\|_2^2 \tag{4a}$$

s.t.

$$E(\boldsymbol{W}) = -\sum_{r=1}^{N}\sum_{i=1}^{n}\{\tilde{y}_{r_i}\ln y_{r_i} + (1-\tilde{y}_{r_i})\ln(1-y_{r_i})\}, \tag{4b}$$

where $\tilde{\boldsymbol{y}}_{\boldsymbol{r}} = \{\tilde{y}_{r_i}\}_{i=1}^{n}$ denotes a target vector, y_{r_i} refers to the i-th element of the activation function output $y(\boldsymbol{x}_{\boldsymbol{r}}, \boldsymbol{W})$, and $\boldsymbol{x}_{\boldsymbol{r}} \in \mathbb{R}^m$ is an input vector. Meanwhile, N is the total number of training data in each batch, and $\odot$ represents the Hadamard product. According to Equation (4), for a large weight value, its corresponding first-order derivative term has a large coefficient. Therefore, it is suitable to improve the network robustness under multiplicative deviation.

During the training, neural weights with SAFs are fixed, while other weights can still be updated in the backward process as follows:

$$\boldsymbol{W}_{t+1} = g(\boldsymbol{W}_t, \eta_t, \frac{\partial \tilde{E}(\boldsymbol{W}_t)}{\partial \boldsymbol{W}_t}), \tag{5}$$

where $\boldsymbol{W}_t$ and η_t are the current weight and the current learning rate, respectively, and $\boldsymbol{W}_{t+1}$ denotes the updated weight. Meanwhile, $g(\cdot)$ refers to the optimizer for updating weights and the learning rate.

We use the adaptive moment estimation (Adam) optimizer [20] to train the network model. In each training iteration, we sample a mini-batch from the training set. Then the SAFs are injected randomly into the current network. A feed-forward calculation is performed on each instance in the mini-batch. We calculate the error on each instance and obtain the accumulated gradient of errors with respect to the weights. Finally, $\boldsymbol{W}_{\boldsymbol{t}}$ is updated based on Equation (5). The above process is repeated until the training iteration number T is reached.

To analyze the neural network's loss behaviors, we adopt the concept of the loss visualization in this work. Based on the loss visualization method [21], we project the loss function of a neural network into 2D hyper-planes, as illustrated in Fig. 3. Note that the center of each plot corresponds to the minimizer, and the two axes parameterize two random directions with normalization. We can see from the figures that

	p_1	p_2	p_3
k_1	0.16	0.12	0.07
k_2	0.05	0.09	0.04
k_3	0.19	0.07	0.05

SA0 sensitivity matrix

	p_1	p_2	p_3
k_1	0.17	0.15	0.18
k_2	0.12	0.08	0.06
k_3	0.11	0.10	0.15

SA1 sensitivity matrix

(a)

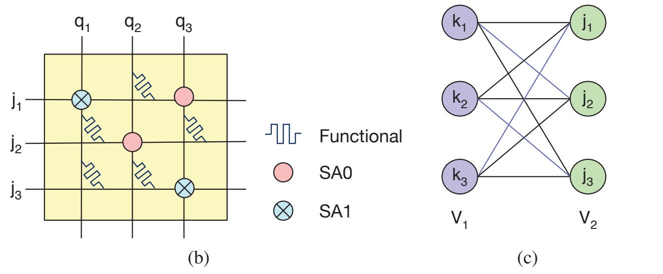


Fig. 4 (a) The sensitivity value of weights $\boldsymbol{W}$ to SAFs; (b) the locations of SAFs in the crossbar $\boldsymbol{C}$; (c) the minimum cost bipartite matching result with $\boldsymbol{V}_r = [2, 3, 1]$ and $\boldsymbol{V}_c = [1, 2, 3]$.

the proposed training method results in a flat minimal and wide contour. The reason is that the proposed general reliability-aware training enhances the network robustness, and thus the loss is not sensitive to faults and variations. Therefore, compared with other training approaches, the proposed method can enhance the network robustness.

V. Chip-Specific Mapping and Re-training

In this section, we present the weight-memristor mapping algorithm. With the generally trained network model and SAF locations in a crossbar as inputs, we propose a chip-specific mapping and re-training to further improve the accuracy loss incurred by SAFs.

During the network training, we can notice that each weight demonstrates different sensitivity to faults. Therefore, when different weights are mapped to SAFs in a crossbar, the influence on computation accuracy of an NCS is not similar. To identify the weights that have large impacts on the computation accuracy, we derive a weight sensitivity measurement as shown in Equation (6):

$$s_{ij}^0 = \frac{|f_i \cdot (w_{ij} - w_{max})|}{\sum_j |f_i \cdot w_{ij}|} \text{ (SA0)}, \quad s_{ij}^1 = \frac{|f_i \cdot (w_{ij} - w_{min})|}{\sum_j |f_i \cdot w_{ij}|} \text{ (SA1)}, \tag{6}$$

where w_{ij} is the weight in i-th row and j-th column of $\boldsymbol{W}$. w_{max} and w_{min} indicate the maximum and minimum values in $\boldsymbol{W}$, which can be modeled as an SA0 fault and an SA1 fault [11]. Meanwhile, f_i represents the i-th element of the input feature vector, and s_{ij}^1 and s_{ij}^0 refer to the sensitivity value of weight w_{ij} to SA1 and SA0, respectively.

Given a generally trained neural network generated by the algorithm described in Section IV-A, we perform the forward process to calculate the sensitivities of each weight to SA1 and SA0 as shown in Equation (6). According to the proposed sensitivity measurement, the weight with a small input or close to SAFs value has a low sensitivity. Thus, adding SAFs into those weights will not dramatically affect the network performance.

To prevent highly sensitive synapses from being mapped to SAFs, a matching-based heuristic algorithm is proposed to establish the mapping relation between the weight matrix and the crossbar. Considering a weight matrix $\boldsymbol{W}$ with the size $m \times n$ and a crossbar $\boldsymbol{C}$ with the same size, we need to determine two vectors that can represent the mapping relation result as follows:

(1) Row mapping vector ($\boldsymbol{V}_r$): $\boldsymbol{V}_r[k] = j$ if the k-th row of $\boldsymbol{W}$ is mapped to the j-th row of $\boldsymbol{C}$ ($1 \le k, j \le m$);

(2) Column mapping vector ($\boldsymbol{V}_c$): $\boldsymbol{V}_c[p] = q$ if the p-th column of $\boldsymbol{W}$ is mapped to the q-th column of $\boldsymbol{C}$ ($1 \le p, q \le n$).

We derive a metric named "summed fault sensitivity (SFS)" to calculate the cost by mapping the k-th row of $\boldsymbol{W}$ to the$j$-th row of $\boldsymbol{C}$ as follows:

Algorithm 1 Matching-based Heuristic

```
Input: $\boldsymbol{W}$, SAF locations in $\boldsymbol{C}$, and sensitivity information.
Output: $\boldsymbol{V}_r$ and $\boldsymbol{V}_c$.
 1: for $i \leftarrow 1$ to $K$ do
 2:     Build a weighted bipartite graph between $\boldsymbol{W}$ and $\boldsymbol{C}$;
 3:     Minimum cost bipartite matching;          ▷ Kuhn_Munkres
 4:     if Solution cost is reduced then
 5:         Update cost and record $\boldsymbol{V}_r$ and $\boldsymbol{V}_c$;
 6:         Break;
 7:     else
 8:         Applying a pairwise column permutation of $\boldsymbol{C}$;
 9:     end if
10: end for
```

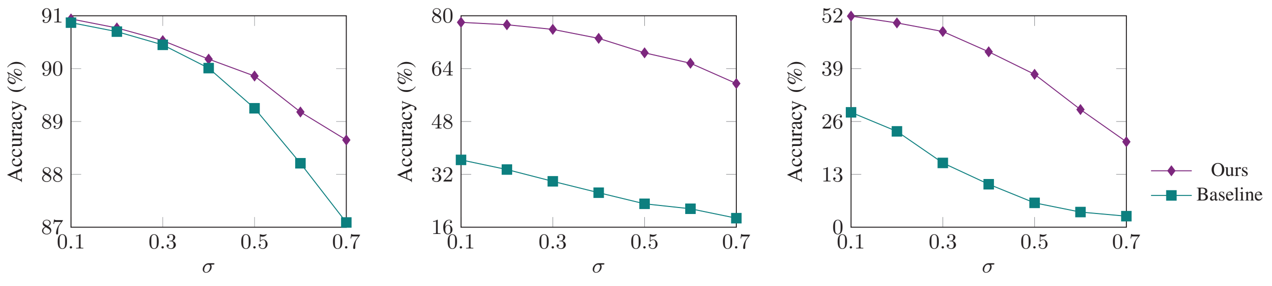
$$SFS_{kj} = \sum_{q=1}^{n} |s_{kq}|, \tag{7}$$

where

$$s_{kq} = \begin{cases} s_{kq}^0, & \text{if } c_{jq} \text{ is an SA0,} \\ s_{kq}^1, & \text{if } c_{jq} \text{ is an SA1.} \end{cases} \tag{8}$$

Algorithm 1 summarizes the algorithm details of the proposed matching-based heuristic algorithm. With the sensitivity measurement and the SAF locations in the crossbar as inputs, the algorithm is expected to generate two vectors. First, we construct a complete bipartite graph $G(V, E)$ to represent all the possible row matching from $\boldsymbol{W}$ to $\boldsymbol{C}$ (line 2). Here the vertex set $V = V_1 \cup V_2$, where V_1 is the row set of $\boldsymbol{W}$ and V_2 is the row set of $\boldsymbol{C}$. In addition, the edge set $E = \{(k, j) | k \in V_1 \text{ is mapped to } j \in V_2\}$. The cost of the edge is calculated by Equation (6). Then, the `Kuhn_Munkres` algorithm is adopted to find a minimum cost bipartite matching [22] in a polynomial time (line 3). The goal in the minimum cost bipartite matching problem is to compute a matching M that minimizes $\sum_{(k,j) \in M} SFS_{kj}$. If the matching cost is improved, we record the current mapping relation (line 5). Otherwise, column permutations are performed for $\boldsymbol{C}$ (lines 8). The above process is repeated until the iteration number K is reached. As a result, highly sensitive synaptic weights are prevented from being mapped to SAFs.

We exemplify the process of the matching-based heuristic algorithm with Fig. 4. For example, assume a weight matrix $\boldsymbol{W}$ with the size 3×3, the sensitivity matrices of weights to SA1 and SA0 are shown in Fig. 4(a). Note that the weight sensitivity values can be calculated by Equation (6). Fig. 4(b) depicts an example of SAF locations in crossbar $\boldsymbol{C}$. Then according to the sensitivity measurement and the SAF locations, a complete weighted bipartite graph between rows of $\boldsymbol{W}$ and rows of $\boldsymbol{C}$ is constructed. Finally a minimum cost matching



(a) MLP on MNIST (b) LeNet on CIFAR-10 (c) AlexNet on CIFAR-100

Fig. 5 Effectiveness of the proposed general reliability-aware training.

TABLE I Original accuracy without considering SAFs and variations.

Network	Dataset	Accuracy
MLP	MNIST	92.8%
LeNet	CIFAR-10	86.2%
AlexNet	CIFAR-100	68.6%

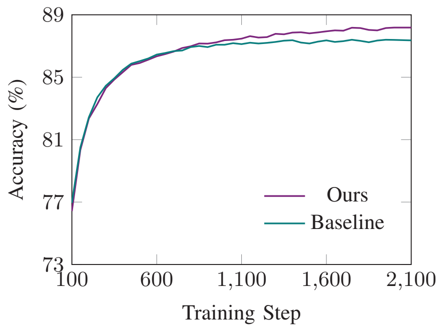


Fig. 6 Training curves of a two-layer MLP on MNIST dataset under 1.0% SAFs and $\sigma = 0.7$.

can be obtained by the `Kuhn_Munkres` algorithm with $\boldsymbol{V}_r = [2, 3, 1]$ and $\boldsymbol{V}_c = [1, 2, 3]$ (blue lines in Fig. 4(c)). It means that rows k_1, k_2 and k_3 of $\boldsymbol{W}$ are mapped to rows j_2, j_3 and j_1 of $\boldsymbol{C}$, while columns p_1, p_2 and p_3 of $\boldsymbol{W}$ are assigned to columns q_1, q_2 and q_3 of $\boldsymbol{C}$.

In addition, after the mapping relation between a weight matrix and a crossbar is determined, the proposed training method is further performed to restore the SAF-caused accuracy loss by re-tuning the weights. Because the weights mapped to SAFs will remain unchanged in re-training iterations, the computation accuracy loss can only be compensated by re-tuning other trainable weights.

VI. Experimental Results

The proposed framework is implemented based on the `Tensorflow` library [23] and validated on a Linux server with an 8-core Intel CPU and an Nvidia Tesla K40M GPU. To verify the effectiveness of our algorithm, we experiment with three datasets, including MNIST [24], CIFAR-10, and CIFAR-100 [25]. First a two-layer multi-layer perception (MLP) with ReLU activation function is trained for MNIST. Then LeNet [24] with two convolutional (Conv) layers and three fully connected (FC) layers is implemented and tested on CIFAR-10. In the convolutional layers, 64 kernels of size 5×5 are employed; three FC layers are consistent, whose dimensions are 384, 192, and 10. Finally, AlexNet [26] is trained on CIFAR-100. We use the 5×5 kernel size to make it suitable for input images. The lognormal distribution is adopted as our memristor variation model, as shown in Equation (1). The SAFs and memristor variation model are injected into the weights across all different layers in the network. According to the published data [11], 83.8% of SAFs are set as SA1 faults and 16.2% of SAFs are fixed on SA0 faults. The performance of the proposed framework is evaluated by a Monte Carlo simulation. TABLE I shows the original computation accuracy of NCS without the impacts of SAFs and variations, which serves as the upper bound of the reliability design.

A. Effectiveness of General Reliability-aware Training

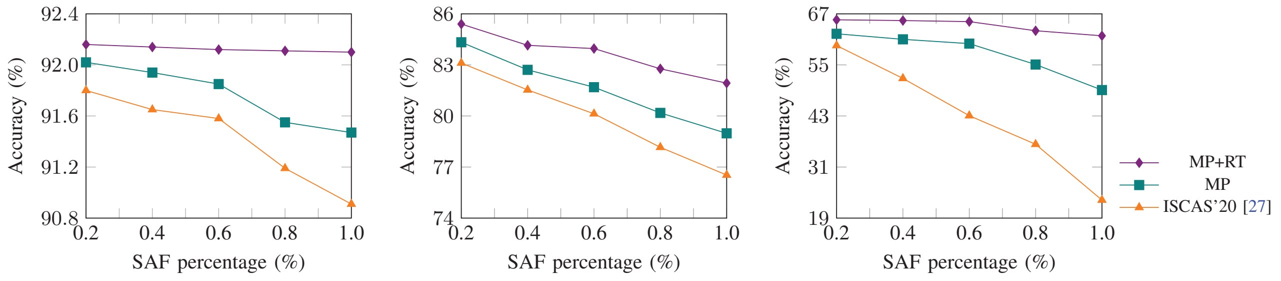
In the first experiment, we demonstrate the robustness of the proposed general reliability-aware training against the SAFs and the device variations. We apply the proposed general reliability-aware training to train the three neural networks. For a comparison, we also employ a traditional learning strategy to train the neural networks (Baseline), where the proposed dropout-inspired approach and the first-order derivative regularizer are not included.

Fig. 5 illustrates the accuracy of the three neural networks derived by the two training methods, respectively. We vary σ to change the influence of stochastic resistance variation. The SAFs percentages are set to 1%. It can be seen from the figure that as σ increases, the traditional learning strategy suffers from severer accuracy degradation. Meanwhile, our general reliability-aware training can significantly improve accuracy, demonstrating the robustness of the training method against SAFs and variations. For example, our training method boosts the accuracy of LeNet_CIFAR-10 from 18.74% to 59.49% even under the significant variation $\sigma = 0.7$, as shown in Fig. 5(b). In addition, the curves of the two training methods on MNIST dataset are depicted in Fig. 6, where x-axis indicates training steps and y-axis is computation accuracy on the testing set. We can see that our general reliability-aware training is a more stable procedure and converges to a higher accuracy.

B. Effectiveness of Mapping and Re-training.

With the generally trained network model as the input, a chip-specific mapping and re-training are further performed to reduce the accuracy loss incurred by SAFs. In [27], a general reliability-driven training scheme is proposed, where the mapping and re-training processes are not contained. In the second experiment, we evaluate the efficiency of the proposed chip-specific mapping (MP) and re-training (RT) against SAFs. Given the trained neural network model, we randomly inject SAFs into weights across all different layers. Then the proposed "MP+RT" is exploited to restore the computation accuracy. Here, "MP" denotes the framework with only mapping, while "ISCAS'20" represents the case without mapping and re-training, which serves as a baseline.

Fig. 7 compares the accuracy of the neural networks after applying the three flows separately. The SAF percentages of MLP_MNIST are chosen between 0.2% and 1.0%. The simulation results indicate that the proposed "MP+RT" can effectively restore the computation



(a) MLP on MNIST (b) LeNet on CIFAR-10 (c) AlexNet on CIFAR-100

Fig. 7 Effectiveness of the proposed mapping (MP) and re-training (RT).

accuracy. As Fig. 7(c) illustrates, for AlexNet_CIFAR-100, the proposed "MP+RT" can improve the accuracy from an unacceptable level (23.24%) to a level close to the original accuracy (61.84%) under 1.0% SAF percentage. A similar trend can also be observed for MLP_MNIST and LeNet_CIFAR-10, as shown in Fig. 7(a) and Fig. 7(b).

VII. Conclusions

In this paper, we have proposed a novel reliability-driven design framework for memristive crossbar-based neuromorphic computing systems, considering both SAFs and variations challenges simultaneously. Experimental results have clearly demonstrated that the proposed method can improve the computation accuracy of NCS and enhance the NCS robustness against SAFs and resistance variations.

VIII. Acknowledgments

This work was partially supported by the National Key R&D Program of China under grant No. 2019YFB2204800, the National Natural Science Foundation of China (NSFC) under grant No. 61904047, 61732020, the Strategic Priority Research Program of Chinese Academy of Sciences under grant No. XDB44000000, the Fundamental Research Funds for the Central Universities under grant No. JZ2019HGBZ0159. The authors would like to thank Information Science Laboratory Center of USTC for the hardware and software services.

References

[1] W. Wen, C.-R. Wu, X. Hu, B. Liu, T.-Y. Ho, X. Li, and Y. Chen, "An EDA framework for large scale hybrid neuromorphic computing systems," in *Proc. DAC*, 2015, pp. 1–6.
[2] Q. Xu, H. Geng, S. Chen, B. Yu, and F. Wu, "Memristive crossbar mapping for neuromorphic computing systems on 3d ic," *ACM TODAES*, vol. 25, no. 1, pp. 1–19, 2019.
[3] Y. Chen, H. H. Li, C. Wu, C. Song, S. Li, C. Min, H.-P. Cheng, W. Wen, and X. Liu, "Neuromorphic computing's yesterday, today, and tomorrow–an evolutional view," *Integration, the VLSI Journal*, vol. 61, pp. 49–61, 2018.
[4] 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. NVMSA*, 2017, pp. 1–6.
[5] W. Huangfu, L. Xia, M. Cheng, X. Yin, T. Tang, B. Li, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Computation-oriented fault-tolerance schemes for RRAM computing systems," in *Proc. ASPDAC*, 2017, pp. 794–799.
[6] Q. Xu, S. Chen, H. Geng, B. Yuan, B. Yu, F. Wu, and Z. Huang, "Fault tolerance in memristive crossbar-based neuromorphic computing systems," *Integration, the VLSI Journal*, vol. 70, pp. 70–79, 2020.
[7] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Handling stuck-at-fault defects using matrix transformation for robust inference of dnns," *IEEE TCAD*, pp. 1–13, 2019, doi:10.1109/TCAD.2019.2944582.
[8] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *Proc. DAC*, 2017, pp. 1–6.
[9] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training with on-line fault detection for rram-based neural computing systems," in *Proc. DAC*, 2017, p. 33.
[10] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: variation-aware training for memristor x-bar," in *Proc. DAC*, 2015, p. 15.
[11] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: learning variations and defects in RRAM crossbar," in *Proc. DATE*, 2017, pp. 19–24.
[12] A. J. Van de Goor and Y. Zorian, "Effective march algorithms for testing single-order addressed memories," *Journal of Electronic Testing*, vol. 5, no. 4, pp. 337–345, 1994.
[13] C.-Y. Chen, H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu, and F. T. Chen, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *Theory of Computing*, vol. 64, no. 1, pp. 180–190, 2014.
[14] A. Asenov, S. Kaya, and A. R. Brown, "Intrinsic parameter fluctuations in decananometer mosfets introduced by gate line edge roughness," *IEEE TED*, vol. 50, no. 5, pp. 1254–1260, 2003.
[15] S. R. Lee, Y.-B. Kim, M. Chang, K. M. Kim, C. B. Lee, J. H. Hur, G.-S. Park, D. Lee, M.-J. Lee, C. J. Kim *et al.*, "Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory," in *Proc. VLSIT*, 2012, pp. 71–72.
[16] T. Liu, W. Wen, L. Jiang, Y. Wang, and G. Quan, "A fault-tolerant neural network architecture," in *Proc. DAC*, 2019, pp. 1–6.
[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[18] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.
[19] C.-T. Chiu, K. Mehrotra, C. K. Mohan, and S. Ranka, "Modifying training algorithms for improved fault tolerance," in *Proc. IJCNN*, vol. 1, 1994, pp. 333–338.
[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[21] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Advances in Neural Information Processing Systems*, 2018, pp. 6389–6399.
[22] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.
[23] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, 2016, pp. 265–283.
[24] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[25] A. Krizhevsky, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.
[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1097–1105.
[27] J. Wang, Q. Xu, B. Yuan, S. Chen, B. Yu, and F. Wu, "Reliability-driven neural network training for memristive crossbar-based neuromorphic computing systems," in *Proc. ISCAS*, 2020, pp. 1–4.