Reliability-Driven Memristive Crossbar Design in Neuromorphic Computing Systems

Qi Xu, *Member, IEEE*, Junpeng Wang, Bo Yuan, *Member, IEEE*, Qi Sun, *Graduate Student Member, IEEE*, Song Chen, *Member, IEEE*, Bei Yu, *Member, IEEE*, Yi Kang, *Member, IEEE*, and Feng Wu, *Fellow, IEEE*

***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, the memristor suffers from the process variations, causing deviation of the actual programming resistance from its target resistance. In this paper, we propose a 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 dropconnect-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 improve computation accuracy loss incurred by SAFs. Experimental results show that the proposed method can boost the computation accuracy of NCS and improve the NCS robustness.**

***Note to Practitioners*—This work is motivated by the manufacturing reliability problem in a memristive crossbar-based NCS. To enhance the robustness of an NCS to SAFs and device variations, this paper presents a reliability-driven design framework with taking account of both general and chip-specific design optimizations. The experimental results have demonstrated that the proposed framework is superior to the prior arts, and can be easily integrated with existing industrial hardware-based fault tolerance solutions for higher accuracy at lower overhead. Memristive crossbar-based computing system gives hope for the anticipated efficient implementation of artificial neuromorphic networks. With the help of the reliability-driven designs, the computation accuracy is restored, and hence we can expect the wide use of memristive crossbar-based computing system in neuromorphic computing applications.**

***Index Terms*—Neuromorphic computing system, memristive crossbar, fault tolerance, robustness.**

Manuscript received 23 September 2020; revised 14 May 2021 and 12 September 2021; accepted 18 October 2021. Date of publication 12 November 2021; date of current version 6 January 2023. This article was recommended for publication by Associate Editor Y. Chen and Editor S. Hu upon evaluation of the reviewers' comments. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61904047, Grant 61874102, Grant 61732020, and Grant 61931008; in part by the Guangdong Provincial Key Laboratory under Grant 2020B121201001; in part by the Chinese Academy of Sciences (CAS) Project for Young Scientists in Basic Research under Grant YSBR-029; and in part by the Strategic Priority Research Program of the CAS under Grant XDB44000000. *(Corresponding author: Song Chen.)*

Qi Xu and Junpeng Wang are with the School of Microelectronics, University of Science and Technology of China, Hefei, China.

Bo Yuan is with the Guangdong Provincial Key Laboratory of Brain-Inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China.

Qi Sun and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR.

Song Chen, Yi Kang, and Feng Wu are with the School of Microelectronics, University of Science and Technology of China, Hefei, China, and also with the Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei, China (e-mail: songch@ustc.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TASE.2021.3125065.

Digital Object Identifier 10.1109/TASE.2021.3125065

I. Introduction

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

Despite of these tremendous advantages, NCS implementations on memristive crossbars encounter some reliability challenges. First, memristors suffer from stuck-at faults (SAFs) which make the memristor stuck at high or low resistance state, leading to a significant yield loss in NCS. Second, in a memristor-based NCS, programming the resistance of the memristors induces stochastic device variations [4]. As a result, the actual programming resistance is deviated 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. Xia *et al*. [5] presented a numerical iteration algorithm to map the weight matrix to the crossbar. Huangfu *et al*. [6] proposed a mapping algorithm with inner fault-tolerance. In order to provide the robustness, two (or multiple) parallel rows in each memristive crossbar were adopted to represent each row in the weight matrix. Nevertheless, the use of additional rows results in high hardware overheads. Xu *et al*. [7] introduced an integer linear programming (ILP)-based algorithm to derive a weight-memristor mapping for fault tolerance. Currently, machine learning techniques have been successfully utilized to address SAFs problem. Liu *et al*. [8] presented a re-training based method to tolerate SAFs. By using the sparsity of neural networks, Xia *et al*. [9] proposed a fault tolerant on-line training with re-mapping techniques. But only the neuron permutation in fully connected layers are considered in the re-mapping method. A set-based processing element architecture was proposed by Chaudhuri *et al*. [10] to tolerate both SAFs and undefined state faults in binary memristor cells. The architecture can improve the fault tolerance of the memristive crossbar-based NCS on applications of binary pattern-matching and handwritten digit recognition. But the design could require high hardware overheads for multi-bit weight mapping. Zhang *et al*. [11] developed a matrix transformation framework to handle the SAFs. However, the solution only focuses on SAFs, overlooking the programming variations.

In addition, to address stochastic device variations, Liu *et al*. [12] developed a variation tolerant training by adjusting the training goal according to the impact of the 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*. [13] investigated a fault and variation tolerant framework for memristive crossbar-based NCS. It first finds an optimal mapping between weights and memristors for SAF and variation tolerance. Then the re-training process based on conventional gradient descent technique is further adopted to tune weights. But the resistance variation values of memristors are known in advance, thus the method is essentially no different from the SAFs-aware approaches. Moreover, when the variation model changes, inference accuracy of the network is degraded, which means the above variation tolerant methods lack robustness to variations. Recently, He *et al*. [14] presented a digital SAF error correction method and a noise injection adaption methodology to overcome SAF effects and current drifts. In addition, Liu *et al*. [15] proposed a collaborative logistic classifier with error correction output code (ECOC) to enhance the network stability. Given any DNN model, the original softmax classifier in the output layer will be replaced by a certain number of collaborative logistic classifiers. Then the SAFs and variations are injected in the output layer, and the weights of collaborative logistic classifiers will be fine-tuned based on the codeword list. But extra hardware circuits are required to collect the output before the output layer.

Although recent works have investigated the SAFs and variations tolerance, their techniques only focus on the chip-specific design of the memristive crossbar-based NCS. Namely, the SAFs tolerant solutions can be derived only if the exact locations of SAFs in crossbar are known in advance. Due to the high testing overhead, a general reliability design of the memristive crossbar-based NCS without the prior testing is required. Besides, despite a general SAFs and variation tolerant design of NCS is presented in the conference version [16], the method cannot be applied to the chip-specific reliability design. Fundamentally different from these approaches, we propose a reliability-driven design framework for a memristive crossbar-based NCS in combination with general and chip-specific design optimization. Key technical contributions of this work are listed as follows.

- We propose a general reliability-aware training scheme to enhance the robustness of NCS to SAFs and device variations. A dropconnect-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 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 accuracy loss incurred by SAFs. We analyze the sensitivity of weights to SAFs in the specified neural network. Based on the sensitivity measurement and the SAF locations in the crossbar, a weight-memristor mapping based on neuron permutation is adopted to prevent the highly sensitive synapses from being mapped to the fault 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 NCS to SAFs and device variations, but also boosts the computation accuracy of NCS.

TABLE I summarizes the differences between the previous works and the proposed framework. The remainder of this paper is organized as follows. Section II presents the preliminary and introduces the problem to be addressed in the 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 VII presents experimental results, followed by conclusion in Section VIII.

II. Preliminaries

A. Fault and Variation Models in Memristors

Due to the immature fabrication technology, reliability is still a major concern in a memristive crossbar-based NCS. Process variations and manufacturing defects in memristor cells have been investigated and classified based on the criticality of their impact on the cell write time [17]. Such non-ideality may lead to stuck-at-one (SA1) fault and stuck-at-zero (SA0) fault, where a cell's resistance cannot be changed with any electrical stimulus. By injecting faults into a fault-free memristor model, SAFs can be simulated. For example, if a memristor cell is unable to switch to its resistance state with an appropriate write pulse, we conclude that an SAF

TABLE I
SUMMARY OF TECHNIQUES USED TO HANDLE SAFs AND DEVICE VARIATIONS IN RELATED WORKS

Works	Fault/Variation model	Fault locations	Mapping cost metric	Permutation method	Mapping algorithm	Re-training strategy	Hardware cost
ASPDAC'17 [6]	only SAFs	known	absolute linear		directly mapping		ineffective
TCAD'19 [9]	SAFs/lognormal variation	known	binary	neuron	heuristic		ineffective
TCAD'20 [11]	only SAFs	known	weighted square	neuron	heuristic	gradient descent	ineffective
DAC'15 [12]	SAFs/lognormal variation	known	absolute linear	router	greedy		ineffective
DATE'17 [13]	SAFs/lognormal variation	known	absolute linear	router	heuristic	gradient descent	ineffective
Ours	SAFs/lognormal variation	known or unknown	absolute weighted linear	neuron	heuristic	gradient descent	effective

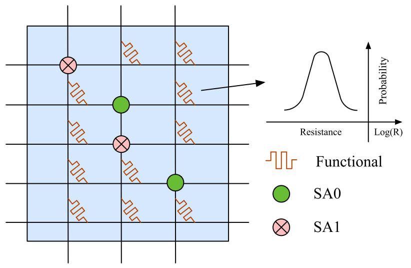


Fig. 1. The memristive crossbar with SAFs and device variations.

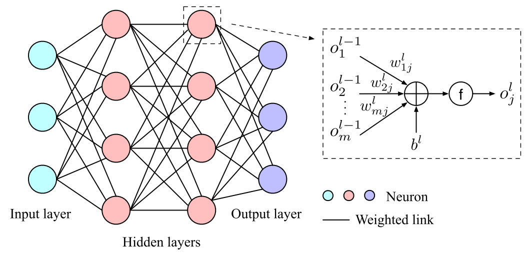


Fig. 2. General representation of deep neural network.

has occurred. Besides, manufacturing defects in metal electrodes include breaks, voids, and oxide-pinholes [18]. Through closed-form expressions that estimate the impact of defects on the memristor's contact resistance, a quantitative analysis of the manufacturing defects can be achieved [17].

The SAF locations in crossbar can be identified by March-C or squeeze-search based testing methods [19], [20], However, the testing overhead may be too high for the NCS implemented by many memristive crossbars. To improve the testing efficiency, Kannan *et al.* [21] exploited sneak-paths inherent in crossbar to test multiple memristors at once. Recently, Chaudhuri *et al.* [22] presented an efficient framework to analyze the functional criticality of SAFs in systolic array. A scalable method based on machine-learning (ML) and generative adversarial network (GAN) was developed to classify SAFs in terms of their functional criticality. The results demonstrated that the proposed scheme can significantly reduce test escapes during the evaluation of fault criticality. In this work, we mainly focus on the SAF tolerance design, thus the above methods can be directly used to classify SAFs in specific memristive crossbars.

In addition, multiple write/read operations strongly drive vacancies in one direction inside the dielectric. Thus, when an external potential difference of opposite polarity is applied, the vacancy will need more time to return. This phenomenon will cause a degradation in memristor operation and hence is viewed as parametric variation [21]. Meanwhile, the driving circuit design can trigger a cycle-to-cycle switching variation [23]. As a result, any small fluctuations in the magnitude and pulse-width of the programming current or voltage can produce resistance variations. Throughout this paper, we use matrix $\boldsymbol{C}$ to represent the resistance states of the crossbar. The work in [24] have reported that the programming resistance states of the memristors follow a lognormal distribution as Equation (1):

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

where $\tilde{c}_{ij}$ and c_{ij} are actual resistance and target resistance of a memristor in i-th row and j-th column of $\boldsymbol{C}$, θ_{ij} denotes the zero-mean Gaussian variation with a variance of σ. Thus, the memristor resistance variation is a multiplicative deviation [25]. An example of a memristive crossbar with SAFs and variations is shown in Fig. 1.

B. Deep Neural Network

A deep neural network (DNN) is a directed acyclic graph comprising of multiple computation layers [26]. A higher level abstraction of the input data or the feature map is extracted from the input layer to the output layer. The intermediate layers between the input and the output layer are called hidden layers. An example of DNN is shown in Fig. 2.

DNNs have two operating processes: training and inference. During training, the neurons in each layer take the intermediate outputs of the previous layer as inputs, as shown in Fig. 2. The neuron output is derived by firstly applying a dot product between its inputs and parameters, i.e., the weights and the bias, and then applying a non-linear activation function to generate the final computation result. Formally, the output of j-th neuron in layer l, o_j^l, is calculated as:

$$o_j^l = f(\sum_i w_{ij}^l \cdot o_i^{l-1} + b^l), \tag{2}$$

where o_i^{l-1} is the output of i-th neuron in the previous layer $l-1$, w_{ij}^l denotes the weights from the i-th neuron in layer $l-1$ to the j-th neuron in layer l, and b^l is the bias in layer l. Meanwhile, $f(\cdot)$ represents the non-linear activation function. In multi-class classification task, a softmax function is usually adopted as the activation function. Formally, the softmax function estimates the probability of a given input pattern belonging to class i as follows:

$$\text{softmax}(i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \tag{3}$$

where z_i is the output of i-th neuron in the output layer. Since the softmax activation function can derive the probability distribution over classes, the class with largest probability value is chosen as the final predicted class.

In addition, in the inference phase, the previously trained DNN model is adopted to predict labels for new input data. Thus the accuracy of trained models can be obtained by performing inference on the testing set. For a more thorough discussion of DNN, please refer to [27].

C. Problem Formulation

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

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

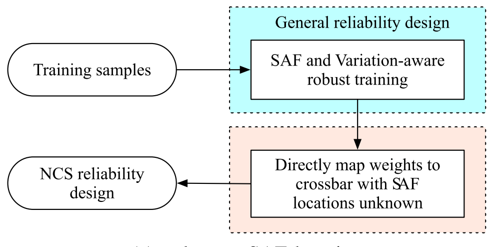
Definition 2 (Computation Accuracy): The computation accuracy of NCS is given by the probability that the trained NCS can successfully classify the test samples.

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

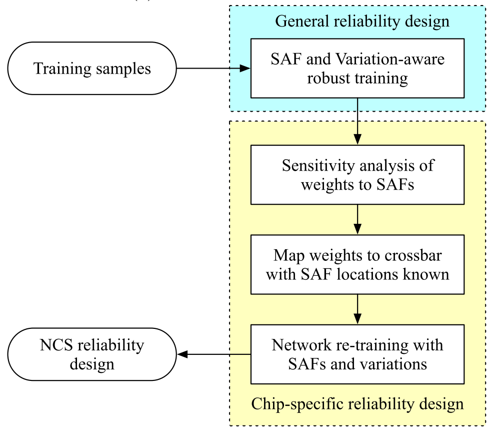
Problem 1 (FVTN): Given the datasets and the device variation model, we implement a neural network on the memristive crossbar with compensation for the impact of faults and device variations, so that the computation robustness and the computation accuracy of NCS are improved.

III. Overall Flow

Fig. 3 illustrates the proposed reliability-driven design framework for NCS, which is developed for two different cases, namely, with and without knowing SAF locations in memristive crossbars. We first propose a general reliability-aware training scheme for NCS, as depicted in Fig. 3(a). Through a dropconnect-inspired technique and a new weighted error function, the robustness of NCS to SAFs and device variations is enhanced. As a result, we can directly map the trained model to memristive crossbar without knowing SAF locations. In addition, a chip-specific design is further proposed to restore the computation accuracy loss incurred by SAFs, as displayed in Fig. 3(b). Based on the neural network model generated by the reliability-aware training,


General reliability design
Training samples
SAF and Variation-aware robust training
NCS reliability design
Directly map weights to crossbar with SAF locations unknown

(a) unknown SAF locations


General reliability design
Training samples
SAF and Variation-aware robust training
Sensitivity analysis of weights to SAFs
Map weights to crossbar with SAF locations known
NCS reliability design
Network re-training with SAFs and variations
Chip-specific reliability design

(b) known SAF locations

Fig. 3. The proposed reliability-driven design framework for NCS.

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 then 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 to variations. Finally, a reliability design of NCS deployed on a memristive crossbar with SAF locations known is achieved.

IV. Reliability-Driven Network Training

In this section, we first introduce our general reliability-aware training algorithm. Then we discuss several other regularizer-based training approaches. By the end, we analyze the loss behaviors of the neural network.

A. General Reliability-Aware Network Training

In the general reliability-aware training scheme, a dropconnect-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 network is trained in software. Since we do not focus on specific memristive crossbars during training, the quantization process is not performed. Thus, the network weights are represented as 32-bit floating-point precision in the general training stage. After the network training, the weights are quantized to 8-bit value, and then mapped to the specific memristive crossbars with SAFs and device variations. According to the results in [28], the accuracy loss of 8-bit quantization is less than 1%.

To prevent complicated dependencies between weights and make the network robust against SAFs, a dropconnect-inspired technique is proposed. We train the network with mini-batch gradient descent (MGD) [29] approach which can provide more efficient computation than stochastic gradient descent (SGD). A group of instances are randomly picked from training set in each training iteration. For each batch, we sample a network with random SAFs distribution. The weights injected with SA0 or SA1 faults are temporarily set to the maximum or minimum values of the current layer where the weight is located. 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 dropconnect strategy in deep learning [30], where the weights 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 prevented.

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 Equation (4):

$$\tilde{E}(\boldsymbol{W}) = E(\boldsymbol{W}) + \lambda\Omega, \tag{4}$$

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) and regularization penalty $\boldsymbol{\Omega}$.

Two popular regularizers are l_1-norm regularizer and l_2-norm regularizer, as defined in Equations (5) and (6):

$$\Omega_{l_1}(\boldsymbol{W}) = \|\boldsymbol{W}\|_1, \tag{5}$$
$$\Omega_{l_2}(\boldsymbol{W}) = \|\boldsymbol{W}\|_2^2. \tag{6}$$

In order to avoid confusion, all norms $\|\cdot\|$ are calculated with respect to flattened vectors. L_1-norm regularizer has the property that if λ is sufficiently large, some weights will be driven to zero, leading to a sparse network model. Compared with l_1-norm regularizer, l_2-norm regularizer gives more bias to the large weights and shrinks the weight distribution to a small value range [31]. To compensate for the impact of faults and device variations, a smooth error curve is needed. According to the work [32], l_2-norm can reduce the network sensitivity and thus enhance the training robustness. Besides, 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. Thus, 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. Considering a large weight case, although the first-order derivative term itself can reflect the information on how the error changes as the weight is perturbed slightly, multiplying the weight with it will impose a large coefficient on the first-order derivative term. Hence the disturbance of large weights can be avoided to a certain extent. In accordance with the argument, we derive our objective function by combining Equations (4) and (6) and the first-order derivative term as:

Algorithm 1 General Reliability-Aware Network Training

```
Input: Training set, $\boldsymbol{W}_t$ and $\eta_t$.
Output: $\boldsymbol{W}_{t+1}$ and $\eta_{t+1}$.
1: for $i \leftarrow 1$ to $T$ do
2:     Sample a mini-batch $B_i$ from training set;
3:     Inject randomly distributed SAFs into network;
4:     Calculate error $\tilde{E}(\boldsymbol{W}_t)$;                    ▷ Equation (7)
5:     Obtain the accumulated gradient of error $\frac{\partial \tilde{E}(\boldsymbol{W}_t)}{\partial \boldsymbol{W}_t}$;
6:     Update $\boldsymbol{W}_t$;                                        ▷ Equation (8)
7:     Restore the weights with SAFs to the original value;
8: end for
```

$$\begin{aligned}\min_{\boldsymbol{W}}\ & E(\boldsymbol{W}) + \lambda_1\|\boldsymbol{W}\|_2^2 + \lambda_2\left\|\boldsymbol{W}\odot\frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{W}}\right\|_2^2\\ \text{s.t. } & E(\boldsymbol{W}) = -\sum_{r=1}^{N}\sum_{i=1}^{n}\{\tilde{y}_{r_i}\ln y_{r_i}\\ &\quad +(1-\tilde{y}_{r_i})\ln(1-y_{r_i})\},\end{aligned} \tag{7}$$

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 activation function output $y(\boldsymbol{x_r}, \boldsymbol{W})$, and $\boldsymbol{x_r} \in \mathbb{R}^m$ 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 (7), for the 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:

$$\boldsymbol{W}_{t+1} = g(\boldsymbol{W}_t, \eta_t, \frac{\partial \tilde{E}(\boldsymbol{W}_t)}{\partial \boldsymbol{W}_t}), \tag{8}$$

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

The details of the proposed general reliability-aware training scheme are illustrated in Algorithm 1. We use the adaptive moment estimation (Adam) optimizer [33] to train the network model. In each training iteration, we sample a mini-batch from the training set (line 2). Then the randomly distributed SAFs are injected into the current network (line 3). A feed-forward calculation is performed on each instance in the mini-batch. We calculate the error on each instance (line 4) and obtain the accumulated gradient of errors with respect to the weights (line 5). Next, $\boldsymbol{W_t}$ is updated based on Equation (8) (line 6). Finally, the weights with SAFs will be restored to the original value (line 7). The above process is terminated until satisfying the training iteration number T.

Fig. 4 illustrates the weight distributions of neural network trained by three different regularizers. A two-layer multi-layer perception (MLP) on MNIST dataset is shown as example. As demonstrated in Fig. 4, compared with the training strategy

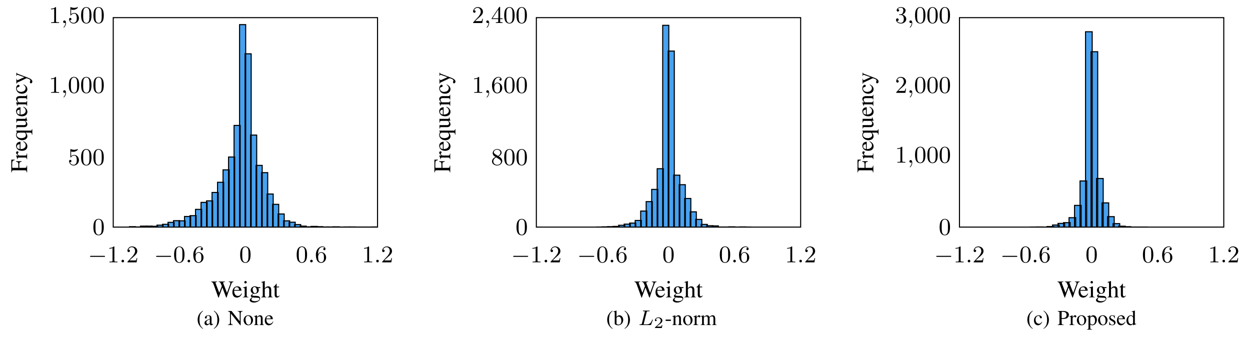


Fig. 4. Weight distributions under different regularizers.

without any regularizers, the proposed regularizer in Equation (7) can constrain the network weight distribution to a small range. Besides, in contrast to the l_2-norm regularizer, the proposed counterpart shrinks the values of the large weights. For the multiplicative resistance variation, the deviations in weights are proportional to weight magnitudes. Therefore, the proposed regularizer can improve the network robustness.

B. Differences With Existing Works

A robust neural network should satisfy the following conditions [37]: (i) be as simple as possible; (ii) have a smooth error curve. In order to meet the above conditions, we propose a new regularizer to train the network as shown in Equation (7). First, the l_2-norm term shrinks the network weight distribution to a small value range. As a result, the generalization performance of the network is improved. Meanwhile, the first-order derivatives term can avoid sharp changes in error with minor changes in weights, which in turn forms smoothness of the learned function. Therefore, the proposed regularizer-based training can enhance the network robustness in a subtly different way. In the following section, we briefly discuss several other regularizer-based training approaches.

As mentioned in Section IV-A, the l_1-norm regularizer helps to realize a sparse network. Note that, if a network model is sparse, the simplicity of the network can somewhat be achieved [37]. Therefore, Liu *et al.* [38] adopts the l_1 regularization to enhance the inherent fault tolerance capability of the network. However, it is obvious to see that the l_1-norm regularizer is not differentiable. This may lead to difficulties on both theoretical analysis and numerical simulations, when the weights are very close to zero.

Since the lower the value of the weight, the higher the fault tolerance of the network. To restrict the weight magnitude, Chiu *et al.* [32] and Zhang *et al.* [34] applied the l_2-norm regularizer to the empirical error. Note that the l_2-norm regularizer is also known as weight decay [31]. By injecting SAFs into network during training, the methods can effectively tolerate SAFs. However, the l_2-norm regularizer-based training approaches cannot address the multiplicative device variation.

To measure the curvature of a solution surface, an output-sensitivity metric (OS) is defined in [35].

$$OS = \sum_{r=1}^{N}\sum_{i=1}^{n}\left\|\frac{\partial y_{r_i}}{\partial \boldsymbol{W}}\right\|_2^2, \tag{9}$$

where N is the total number of training data in each batch, y_{r_i} refers to the i-th element of the output vector $y(\boldsymbol{x_r}, \boldsymbol{W})$, and $\boldsymbol{x_r} \in \mathbb{R}^m$ denotes an input pattern. Specifically, the lower the output-sensitivity, the higher the network stability against the variation. Based on the measurement, Edwards and Murray [35] and He *et al.* [36] appended the output-sensitivity term to the error function. Consequently, the network output is not sensitive to the weight variations. As defined in Equation (9), the output-sensitivity considers the derivatives of the output with respect to the weights, whereas the derivatives of the error with respect to the weights are employed in the proposed regularizer as in Equation (7). Note that $(\partial E/\partial w_{ij})^2 = e_j^2 \cdot (\partial o_j/\partial w_{ij})^2$, where w_{ij} is a weight, and e_j and o_j are the error and the output of the j-th neuron in the output layer. Considering the general case when neither e_j nor $(\partial o_j/\partial w_{ij})$ is too small, if $(\partial E/\partial w_{ij})$ is small, both of e_j and $(\partial o_j/\partial w_{ij})$ should be small. Thus, minimization of $(\partial E/\partial w_{ij})$ will result in minimization of both e_j and $(\partial o_j/\partial w_{ij})$. However, only minimization of $(\partial o_j/\partial w_{ij})$ in Equation (9) does not promote the reduction of e_j. As a result, the performance of network may even be degraded due to the influence of the regularizer. In addition, the output-sensitivity regularizer in [35] and [36] is not proportional to weight magnitudes, the regularizer cannot deal with the multiplicative variation.

As stated in [39], the second-order Tikhonov regularizer provides the lowest order of smoothness. Following the argument, the second-order derivatives of the error function with respect to weight can help to find a solution with a smoother error surface. Besides, for the multiplicative resistance variation, the deviations in weights are proportional to weight magnitudes. That is a large weight value would cause a large weight deviation for the multiplicative variation. In consideration of the two factors, Bernier *et al.* [40] developed an error function as Equation (10).

$$\min_{\boldsymbol{W}} \; E(\boldsymbol{W}) + \lambda \sum_{i=1}^{m} \boldsymbol{w}_i^\top \boldsymbol{H}_i \boldsymbol{w}_i \tag{10a}$$

$$\text{s.t. } \boldsymbol{H}_i = \operatorname{diag}\{\frac{\partial^2 E(\boldsymbol{W})}{\partial w_{i1}^2}, \ldots, \frac{\partial^2 E(\boldsymbol{W})}{\partial w_{in}^2}\}, \tag{10b}$$

Here $E(\boldsymbol{W})$ is the mean-square error, while $\boldsymbol{w}_i$ is the i-th row vector of weight matrix $\boldsymbol{W}$. $\boldsymbol{H}_i$ denotes an $n \times n$ diagonal matrix, whose elements are the second derivatives of the error function with respect to $\boldsymbol{w}_i$. By multiplying the

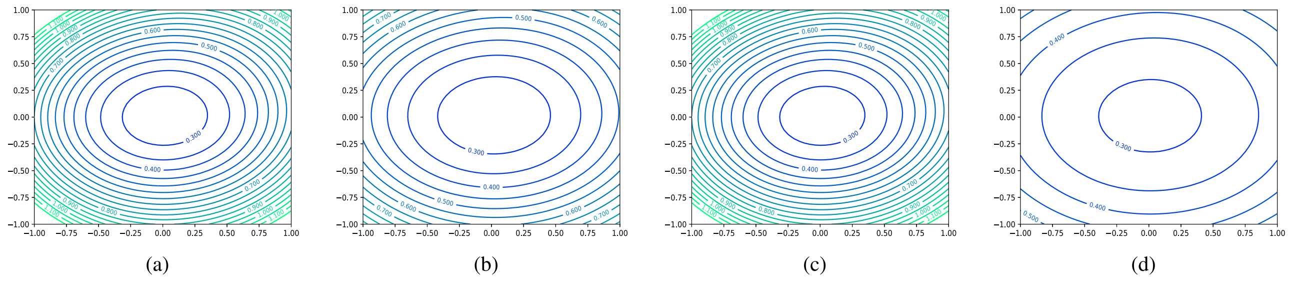


Fig. 5. 2D visualization of the loss surface of a two-layer MLP trained with different regularizers on MNIST dataset: (a) only the cross-entropy error; (b) the l_2-norm regularizer [32], [34]; (c) the output sensitivity-based regularizer [35], [36]; (d) the proposed regularizer.

TABLE II
NOTATIONS USED IN CHIP-SPECIFIC MAPPING

Notations	Description
$\boldsymbol{W}_l$	Weight matrix connecting layer l to layer $l+1$
$\boldsymbol{C}_l$	Memristive crossbar to which $\boldsymbol{W}_l$ is mapped
m_l, n_l	Number of rows and columns (neurons) of $\boldsymbol{W}_l$
w^l_{ij}	Weight value in the i-th row and j-th column of $\boldsymbol{W}_l$
w^l_{max}, w^l_{min}	Maximum and minimum values in $\boldsymbol{W}_l$
x^l_i	i-th element of the input feature vector to layer l
$s^l_{ij\_1}$, $s^l_{ij\_0}$	Sensitivity values of weight w^l_{ij} to SA1 and SA0
L	Number of layers in network
M_l	Bipartite matching in layer l
K	Iteration numbers

square of weights and the second derivatives of the error function with respect to the weights, a better fault tolerance against weight perturbation is achieved. However, as illustrated in Equation (10), the regularizer is a sum of the second-order derivatives which may have positive or negative sign. Consequently, it may happen that although some second-order derivatives are large, the regularizer term in Equation (10) is small. Hence, a robust neural network cannot be trained in the case. But for the proposed regularizer in Equation (7), this issue does not occur because we use the sum of squares of the derivatives.

To analyze the network's loss behaviors, we adopt the concept of the loss visualization in this work. Based on the loss visualization method [41], we project the loss function of a neural network into 2D hyper-planes, as illustrated in Fig. 5. 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 the proposed training method results in a flat minimal and wide contour, which indicates 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

After the network training, the weights are first quantized to 8-bit value. Then with the generally trained network model and SAFs locations in crossbar as input, we further propose a chip-specific mapping and re-training to improve accuracy loss incurred by SAFs. In this work, each memristive cell

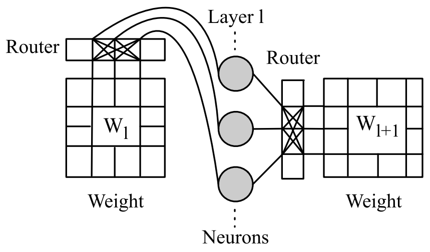


Fig. 6. Row and column permutations using routers in [7] and [44].

maintains 256 quantization levels (8-bit) [42]. As a result, an 8-bit weight can be mapped to one memristor cell. Due to the huge ADC overhead, mapping an 8-bit weight to one memristor cell is impractical [43]. In fact, the proposed chip-specific mapping method can also handle the case where a multi-bit weight is mapped to multiple memristor cells, as described in Section VI. For convenience, some notations used in this section are listed in TABLE II.

Actually, since a neural network comprises of multiple cascaded layers as shown in Fig. 2, the outputs of a previous memristive crossbar should be connected to the inputs of the next crossbar through neurons. In [7] and [44], a matching-based fault tolerance algorithm is given to derive a weight-memristor mapping. In order to ensure the correctness of the matrix-vector multiplication on memristive crossbar with SAFs, the permutations of rows and columns in weight matrix are performed. However, for each layer's weight matrix with n_l neurons, if we independently re-order the rows or columns, an $n_l \times n_l$ routing module is needed to connect crossbars, which introduces a high area overhead. Fig. 6 illustrates the process of rows or columns permutation using routers. To avoid the use of routers, the neuron permutation is presented in [9] to enable the re-ordering of rows and columns in weight matrix.

Due to the low hardware cost, we adopt the neuron permutation [9] in fully connected layers and feature map permutation [11] in convolutional layers. For a fully connected layer, each column in $\boldsymbol{W}_l$ refers to the weights of the synapses connected to a neuron in layer l. Meanwhile, for a convolutional layer, each column in $\boldsymbol{W}_l$ represents the weights of a kernel in layer l. Therefore, permuting the order of two neurons in fully connected layer l is equivalent to exchange two columns in $\boldsymbol{W}_l$ and the corresponding two rows in $\boldsymbol{W}_{l+1}$, as shown in Fig. 7(a). Note that the rows in $\boldsymbol{W}_l$ and the columns in $\boldsymbol{W}_{l+1}$ are fixed

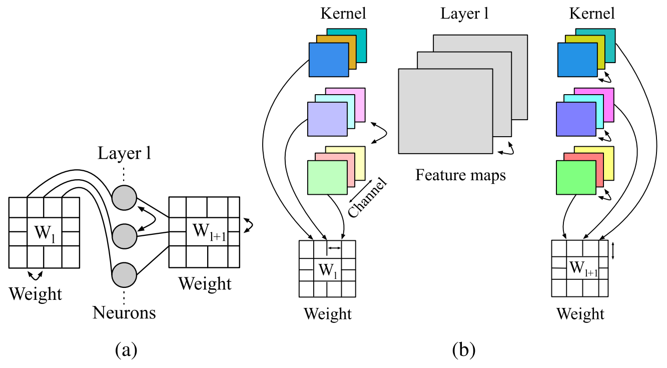


Fig. 7. (a) Neuron permutation in fully connected layer l; (b) feature map permutation in convolutional layer l.

when permuting the neurons in layer l. Similarly, permuting the order of two feature maps in convolutional layer l relates to the re-ordering of two kernels in layer l. Accordingly, the corresponding channels of each kernel in layer $l+1$ are also exchanged, as illustrated in Fig. 7(b). As a result, two columns in $\boldsymbol{W}_l$ and a set of rows in $\boldsymbol{W}_{l+1}$ are re-ordered. The sizes of the set equal to the kernel size. For instance, for a 3×3 kernel, the size of the set is 3×3.9.

During the network training, we can notice that each weight demonstrates different sensitivity to faults. Therefore, when different weights are mapped to SAFs in crossbar, the influence on computation accuracy of an NCS is dissimilar. To identify the weights that have large impacts on the computation accuracy, we derive a weight sensitivity measurement as Equation (11):

$$s^l_{ij\_0} = \frac{\left|x^l_i \cdot (w^l_{ij} - w^l_{max})\right|}{\sum_k \sum_j \left|x^l_k \cdot w^l_{kj}\right|} \text{ (SA0)},$$
$$s^l_{ij\_1} = \frac{\left|x^l_i \cdot (w^l_{ij} - w^l_{min})\right|}{\sum_k \sum_j \left|x^l_k \cdot w^l_{kj}\right|} \text{ (SA1)}. \tag{11}$$

Given the generally trained neural network generated by algorithm in Section IV-A, we perform the forward process to calculate the sensitivities of each weight to SA1 or SA0 as shown in Equation (11). 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 dramatically affect the network performance.

We formulate the neurons or feature maps permutation problem as a minimum cost bipartite matching problem. As shown in Fig. 7, the neuron permutations in fully connected layer are essentially the same as the feature map permutations in convolutional layer. Thus, we only take the neuron permutations into account in following analysis. To prevent the highly sensitive synapses in $\boldsymbol{W}_l$ from being mapped to the SAFs in $\boldsymbol{C}_l$, we assign each neuron to the appropriate one of n_l positions. We first construct a complete bipartite graph $G(V, E)$ as shown in Fig. 8. Here vertex set $V = V_1 \cup V_2$, where V_1 is the set of neurons and V_2 is the set of positions. Besides, the edge set $E = \{(k, j)|k \in V_1$ is mapped to $j \in V_2\}$. Since exchanging the order of two neurons in layer l is equivalent to exchanging the two columns in $\boldsymbol{W}_l$ and the corresponding two rows in $\boldsymbol{W}_{l+1}$, a metric named "summed fault sensitivity (SFS)" is derived to calculate the cost by mapping the k-th neuron to j-th position as follows:

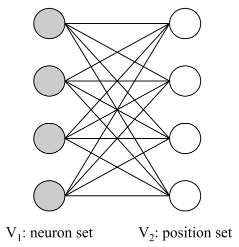


Fig. 8. Complete bipartite graph between neurons and positions.

Algorithm 2 Matching-Based Heuristic

Input: A trained neural network, a set of crossbars with SAFs, and sensitivity information.

Output: Mapping relation between weight matrices and crossbars.

```
1: Initialization: Generate the order of neurons layer by layer;
2: for $i \leftarrow 1$ to $K$ do
3:     Randomly select a layer $l$;
4:     Re-solve the order of neurons in layer $l$;        ▷ Kuhn_Munkres
5:     if Solution cost is reduced then        ▷ Equation (13)
6:         Update the order of neurons in layer $l$;
7:     end if
8: end for
```

$$SFS^l_{kj} = \sum_{p=1}^{m_l}\left|s^l_{pk}\right| + \sum_{q=1}^{n_{l+1}}\left|s^{l+1}_{kq}\right|, \tag{12}$$

where

$$s^l_{pk} = \begin{cases} s^l_{pk\_0}, & \text{if } c^l_{pj} \text{ is an SA0,} \\ s^l_{pk\_1}, & \text{if } c^l_{pj} \text{ is an SA1,} \\ 0, & \text{otherwise.}\end{cases}$$
$$s^{l+1}_{kq} = \begin{cases} s^{l+1}_{kq\_0}, & \text{if } c^{l+1}_{jq} \text{ is an SA0,} \\ s^{l+1}_{kq\_1}, & \text{if } c^{l+1}_{jq} \text{ is an SA1,} \\ 0, & \text{otherwise.}\end{cases}$$

The first term $\sum_{p=1}^{m_l}\left|s^l_{pk}\right|$ is the cost of mapping k-th column of $\boldsymbol{W}_l$ to j-th column of $\boldsymbol{C}_l$. The second term $\sum_{q=1}^{n_{l+1}}\left|s^{l+1}_{kq}\right|$ indicates the cost of assigning k-th row of $\boldsymbol{W}_{l+1}$ to j-th row of $\boldsymbol{C}_{l+1}$. In [9], a binary cost metric is presented to capture whether each weight is realized correctly or not. According to our preliminary observations, compared with the simple metric, the proposed SFS metric can improve classification accuracy.

Since a set of memristive crossbars are required to implement the multiple-layer network, we propose a heuristic algorithm to iteratively optimize the order of neurons over

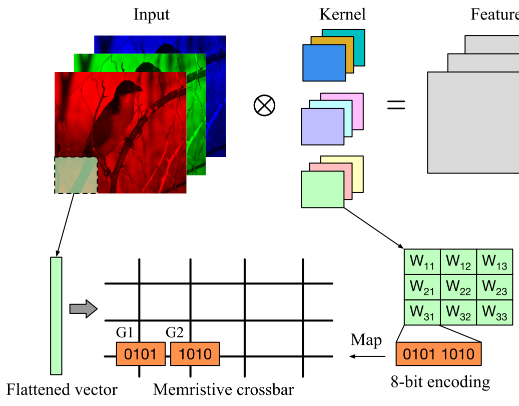


Fig. 9. An example of mapping weights in a convolution layer of DNN to memristive crossbars.

all layers. The objective function of the weight-memristor mapping is defined as Equation (13):

$$\min \sum_{l=1}^{L-1} \sum_{(k,j)\in M_l} SFS_{kj}^{l}. \tag{13}$$

The details of the proposed heuristic algorithm are summarized in Algorithm 2. First, we generate the order of neurons layer by layer, which serves as an initial solution (line 1). Then we randomly select a layer l, and re-solve the order of neurons in the layer by Kuhn_Munkres algorithm [45] (lines 3-4). When permuting the order of the neurons in layer l, the rows in $\boldsymbol{W}_l$ and the columns in $\boldsymbol{W}_{l+1}$ are fixed. If the matching cost in Equation (13) is improved, we update the order of neurons in layer l (line 6). The above process is terminated until satisfying K. As a result, the highly sensitive synaptic weights are prevented from being mapped to SAFs. Since the Kuhn_Munkres algorithm can be solved with time $\mathcal{O}(n_l^3)$ [46], the time complexity of the proposed heuristic is $\mathcal{O}(Kn_l^3)$. The Kuhn_Munkres algorithm is detailed in Appendix .

In addition, after the mapping relation between weight matrix and crossbar is determined, a similar re-training algorithm is further performed to restore the accuracy loss incurred by SAFs [8], [13]. 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. Multiple Memristors Mapping

In this section, we discuss how multi-bit weights are mapped to multiple memristor cells. Instead of representing an 8-bit weight in a single memristor cell as shown in Section V, we can also map one 8-bit weight to $8/\omega$ ω-bit memristor cells located in the same row [43]. Fig. 9 illustrates an example of the process of mapping kernels to a memristive crossbar. As shown in the figure, if the memristor cell has a 4-bit resolution, we use two different memristor cells for complete representation of an 8-bit weight [47]. When an input vector is generated, the memristor cells in a column perform the vector dot-product operation. The results of adjacent two columns must then be merged with the appropriate set of shift and add operations. Similarly, with a 2-bit resolution of each memristor cell, four separate memristor cells are needed to represent the same 8-bit weight [43]. Note that, other layers, e.g., fully connected layers, also involve similar mapping processes [48]. In the subsequent discussion in the section, the memristor cell with 4-bit resolution is considered.

As discussed earlier, mapping network weights to memristive crossbars with SAFs can lead to DNN models with poor accuracy. For instance in Fig. 9, if input vector V is applied to the memristive crossbar under the ideal condition without SAFs, we would get partial outputs O_1 and O_2 corresponding to conductance G_1 and G_2. The partial output can be calculated as $O_i = V \times G_i$. Through the shift and add operations, the final output can be produced according to Equation (14).

$$O = O_1 \times 2^4 + O_2 \times 2^0. \tag{14}$$

Actually, in the non-ideal case, SAFs occur in some cells in memristive crossbar. As shown in Fig. 9, only the two memristor cells to which the 8-bit weight is mapped are all SA1 or SA0 faults, the weight will be represented as the minimum or maximum value. But compared with the low-order bits case, mapping the high-order bits of the weight to SAFs has a higher impact on the accuracy. For example, if G_1 is an SA1 or SA0 fault, the encoding of G_1 changes from 0101 to 0000 or 1111. Depending on the absolute value of the efficient O_1, an error magnified or diminished by a factor of 2^4 is introduced to the output. As a result, a significant accuracy loss is incurred by SAFs.

To restore the accuracy loss, we also adopt the chip-specific mapping method introduced in Section V to handle the case, where a multi-bit weight is mapped to multiple memristor cells. But the weight sensitivity measurement in Equation (11) should be changed to the following equation:

$$\begin{aligned} s_{ijb\_0}^{l} &= \frac{(2^4)^b \left| x_i^l \cdot (w_{ijb}^l - w_{max}^l) \right|}{\sum_k \sum_j \left| x_k^l \cdot w_{kj}^l \right|} \text{ (SA0)}, \\ s_{ijb\_1}^{l} &= \frac{(2^4)^b \left| x_i^l \cdot (w_{ijb}^l - w_{min}^l) \right|}{\sum_k \sum_j \left| x_k^l \cdot w_{kj}^l \right|} \text{ (SA1)}, \end{aligned} \tag{15}$$

where w_{ijb}^l is the b-th group of weight w_{ij}^l $(0 \le b \le 1)$. The two groups store 4-bit weights for the 7··4 and 3··0 segments respectively. The values of w_{max}^l and w_{min}^l are set to 1111 and 0000, which can be modeled as SA0 fault and SA1 fault. Thus Equation (15) measures the sensitivity of mapping each group in a weight to SA0 fault and SA1 fault. Therefore, in order to calculate the cost by mapping the k-th neuron to j-th position, the summed fault sensitivity metric in Equation (12) is adjusted to Equation (16).

$$SFS_{kj}^{l} = \sum_{p=1}^{m_l} \sum_{b=0}^{1} \left| s_{pkb}^{l} \right| + \sum_{q=1}^{n_{l+1}} \sum_{b=0}^{1} \left| s_{kqb}^{l+1} \right| \tag{16}$$

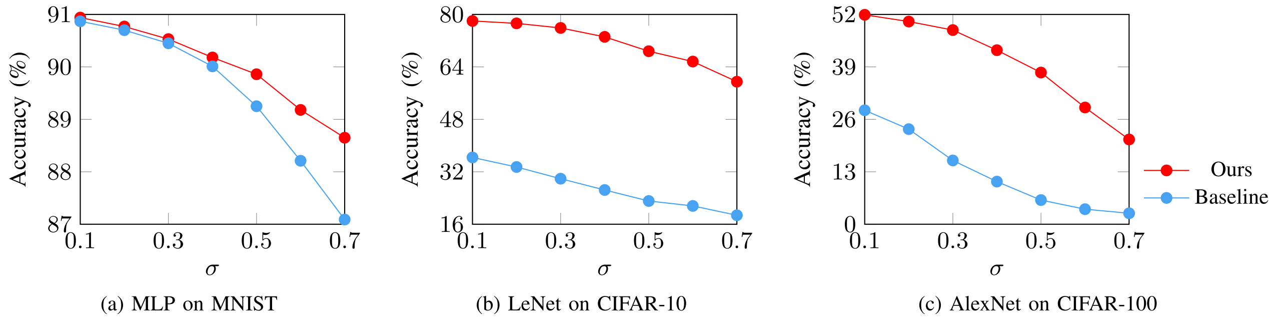


Fig. 10. Effectiveness of the proposed general reliability-aware training.

where

$$s^l_{pkb} = \begin{cases} s^l_{pkb\_0}, & \text{if } c^l_{pjb} \text{ is an SA0,} \\ s^l_{pkb\_1}, & \text{if } c^l_{pjb} \text{ is an SA1,} \\ 0, & \text{otherwise.} \end{cases}$$

$$s^{l+1}_{kqb} = \begin{cases} s^{l+1}_{kqb\_0}, & \text{if } c^{l+1}_{jqb} \text{ is an SA0,} \\ s^{l+1}_{kqb\_1}, & \text{if } c^{l+1}_{jqb} \text{ is an SA1,} \\ 0, & \text{otherwise.} \end{cases}$$

The first term $\sum_{p=1}^{m_l}\sum_{b=0}^{1}\left|s^l_{pkb}\right|$ is the cost of mapping the two groups in k-th column of $\boldsymbol{W}_l$ to the two groups in j-th column of $\boldsymbol{C}_l$. The second term $\sum_{q=1}^{n_{l+1}}\sum_{b=0}^{1}\left|s^{l+1}_{kqb}\right|$ indicates the cost of assigning k-th row of $\boldsymbol{W}_{l+1}$ to j-th row of $\boldsymbol{C}_{l+1}$. Based on the heuristic algorithm 2, the highly sensitive synaptic weights are prevented from being mapped to SAFs.

VII. Experimental Results

The framework is implemented based on `Tensorflow` library [49] and validated on a Linux server with 8-core Intel CPU and Nvidia Tesla K40M GPU. In the general training stage, the network weights are represented as 32-bit floating-point precision. After the network training, the weights are quantized to 8-bit value, and then mapped to the specific memristive crossbars. In the experiment, each memristive cell maintains 256 quantization levels (8-bit) [42]. To verify the effectiveness of our algorithm, we experiment on three datasets, including MNIST [50], CIFAR-10, and CIFAR-100 [51]. First a two-layer multi-layer perception (MLP) with softmax activation function is trained for MNIST. Then LeNet [50] which consists of 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 [52] is trained for CIFAR-100. We use 5×5 kernel size to make it suitable for input images. The lognormal distribution is adopted as our memristor variation model, shown as 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 [13], 83.8% of SAFs are SA1 faults and 16.2% of SAFs are SA0 faults. The performance of the proposed framework is evaluated by a Monte Carlo simulation. TABLE III shows the original computation accuracy of NCS without the impacts of SAFs and variations, which serves as the upper bound of the reliability design.

TABLE III
Original Accuracy Without Considering SAFs and Variations

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

A. Effectiveness of General Reliability-Aware Training

In the first experiment, we demonstrate the robustness of the proposed general reliability-aware training to 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 dropconnect-inspired approach and the first-order derivative regularizer are not included.

Fig. 10 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 1.0% [53]. 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 to 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 $\sigma = 0.7$, shown as in Fig. 10(b). In addition, the curves of the two training methods on MNIST dataset are depicted in Fig. 11, 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 training procedure and converges to a higher accuracy.

B. Effectiveness of Mapping and Re-Training

With the generally trained network model as input, a chip-specific mapping and re-training is further performed to

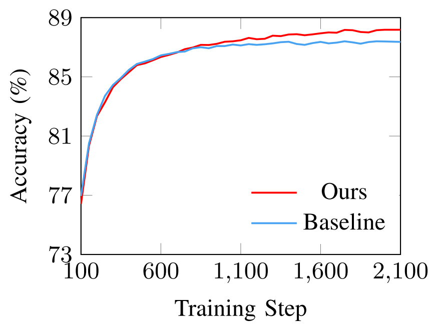


Fig. 11. Training curves of a two-layer MLP on MNIST dataset under 1.0% SAFs and $\sigma = 0.7$.

improve accuracy loss incurred by SAFs. In the preliminary conference version [16], 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 generated by the training scheme in [16], we inject SAFs into weights across all different layers. Then the proposed "MP+RT" is exploited to restore the computation accuracy loss. Besides, "MP" denotes the framework with only mapping, while "ISCAS'20" represents the case without mapping and re-training, which serves as a baseline.

Fig. 12 compares the computation accuracy of the neural networks after applying the three flows respectively. The SAFs percentages are chosen between 0.2% and 1.0% [53]. The simulation results indicate that the proposed "MP+RT" can effectively restore the accuracy. As Fig. 12(c) illustrates, for AlexNet_CIFAR-100, the proposed "MP+RT" can raise the accuracy from an unacceptable value (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. 12(a) and Fig. 12(b).

To demonstrate the scalability of the proposed heuristic algorithm, we further evaluate the performance in the case of mapping multi-bit weights to multiple memristor cells. In the experiment, the memristor cell has a 4-bit resolution. Thus we use two different memristor cells for complete representation of the 8-bit weight. The experiment is run on MLP_MNIST. "Baseline" indicates that the 8-bit network model is directly implemented on the memristive crossbars without the heuristic mapping algorithm. "MP" denotes the case of mapping the multi-bit model to the memristive crossbars by the heuristic method. Fig. 13 shows the computation accuracy of the neural network after applying the above two flows. According to the observation of Fig. 13, the proposed heuristic mapping algorithm can effectively improve the accuracy, which verifies the mapping method can also handle the case where a multi-bit weight is mapped to multiple memristor cells.

In addition, the iteration number K in Algorithm 2 is set through experimental results. Since the time complexity of the proposed heuristic is $\mathcal{O}(Kn_l^3)$, the runtime of the algorithm increases linearly with K value. The experiment is run on LeNet_CIFAR-10. The SAFs percentage is set to 1.0%. We draw the relationship curve between the computation accuracy and K value in Fig. 14. It can be noticed that the accuracy increases along with the increase of K value and finally converges to a stable value. Due to the relatively high accuracy and low runtime, we set K to 20.

C. Comparison With Previous Works

We first compare the proposed general reliability-aware training with three other network training approaches. In Vortex [12], a general variation tolerant training is developed by adjusting the training goal according to the impact of the variations. To restrict the weight magnitude, Chiu *et al*. [32] and Zhang *et al*. [34] applied the l_2-norm regularizer to the empirical error. As a result, all weights have low sensitivity to faults and hence the robustness of network is improved. Besides, Edwards and Murray [35] and He *et al*. [36] adopted the output-sensitivity (OS) regularizer to train the network. Consequently, the influence of weight variations on network output is reduced. The comparison is performed on MNIST dataset by using a two-layer MLP and the results are depicted in Fig. 15. It can be seen that the proposed general reliability-aware training significantly improves the NCS robustness to faults and variations. For example, the proposed reliability-aware training method outperforms Vortex in terms of accuracy by 16.49% on average.

We further compare the proposed chip-specific mapping (MP) with two other mapping-based methods. To eliminate the impact of SAFs, Zhang *et al*. [11] presented a matrix transformation for fault tolerance. In addition, a numerical iteration algorithm is developed in [5] to map the weight matrix to the crossbar. In order to compare with [11], we replace the mapping cost in Equation (12) with the cost metric in [11]. In the experiment, a three-layer MLP for MNIST dataset is implemented, whose size is $784 \times 256 \times 10$. Based on the same trained network model and SAF locations in crossbar, we execute the three mapping strategies respectively. For fair comparisons, the redundancy factor is set to 1 in [11]. Fig. 16 shows the accuracy of the neural network after applying the three methods respectively. The percentages of SAFs are chosen between 2% and 10%. The simulation results indicate that the proposed chip-specific mapping can effectively restore the accuracy. For instance, the proposed chip-specific mapping surpass [11] and [5] in accuracy of 1.64% and 16.79% on average, respectively. Besides, a specific variation tolerant method based on mapping is presented in [13]. However, in reality, the stochastic variation value of memristors in crossbar can hardly be tested in advance.

D. Hardware Overhead Analysis

In this section, we analyze the hardware cost of the proposed chip-specific mapping. As shown in Section V, the neuron and feature map permutations are adopted to establish the mapping relation between weight matrix and the crossbar. Therefore, considering the internal fully connected layer and convolutional layer, the proposed chip-specific mapping introduces no hardware overhead. Meanwhile, for the output layer with softmax activation function, we will allocate a

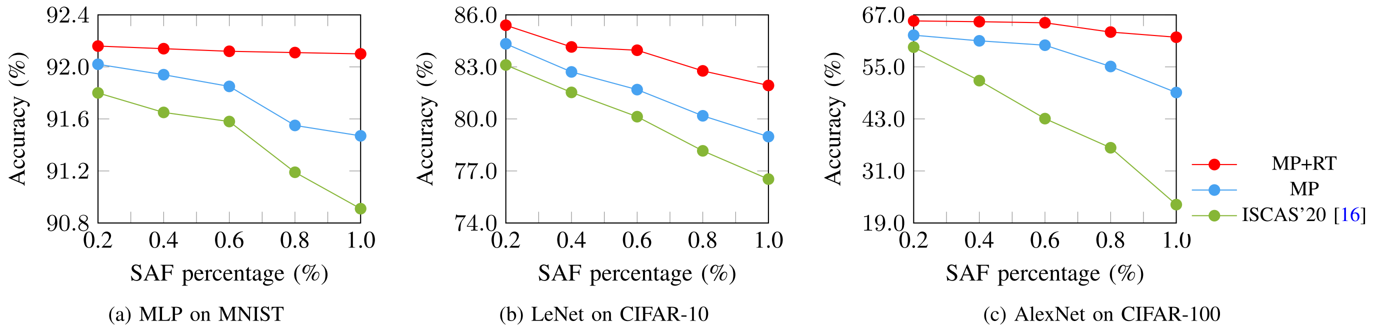


Fig. 12. Effectiveness of the proposed mapping (MP) and re-training (RT).

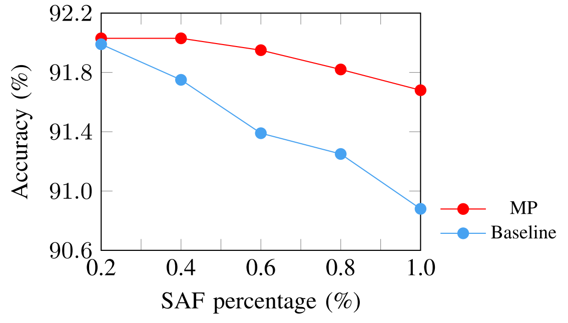


Fig. 13. Effectiveness of the proposed heuristic algorithm on multiple memristors mapping.

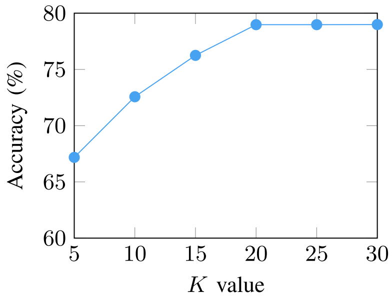


Fig. 14. Effect of the iteration number K on computation accuracy.

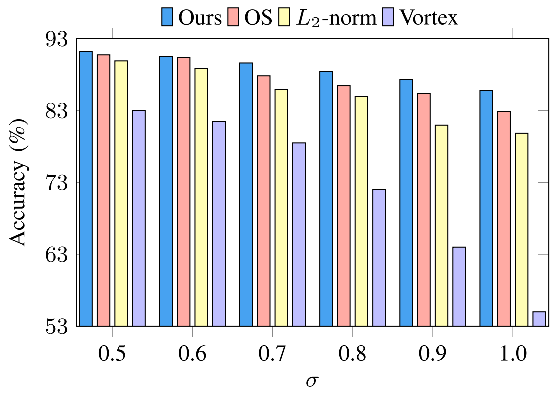


Fig. 15. Comparison of different network training approaches. Items "OS" and "L_2-norm" list the accuracy of the output sensitivity-based training methods [35], [36] and the l_2-norm regularizer-based training approaches [32], [34], while "Vortex" represents the accuracy result in [12].

10-to-1 multiplexer to each neuron to perform permutation. Thus, the hardware overhead of the chip-specific mapping can be neglected. On the other hand, the row and column

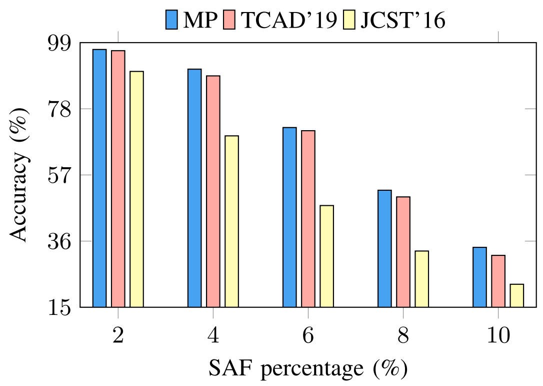


Fig. 16. Comparison of different mapping strategies, where "MP" and "TCAD'19" illustrate the accuracy results of the proposed chip-specific mapping and the matrix transformation framework in [11], while "JCST'16" denotes the result of the numerical iteration mapping algorithm in [5].

permutations are performed in [7] and [44] to derive a weight-memristor mapping. However, since the outputs of a crossbar are connected to the inputs of another crossbar, if we independently re-order the rows or columns, a routing module is needed to connect different crossbars as shown in Fig. 6. As a result, a large hardware cost is introduced.

VIII. Conclusion

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

Appendix

We provide a summary of `Kuhn_Munkres` algorithm as follows.

Step 1: Create a cost matrix with the size of $n_l \times n_l$, where each element represents the cost of mapping one of n_l neurons to one of n_l positions. For each row of the matrix, subtract its smallest element from every element in the row.

Step 2: For all zeros z in the resulting matrix, label z as a starred zero if there is no starred zero in its row or column.

Step 3: Cover each column containing a starred zero. If n_l columns are covered, a complete set of unique mapping is generated according to the positions of starred zeros (exit). Otherwise, go to step 4.

Step 4: Choose an arbitrary non-covered zero and prime it. If there is no starred zero in the row containing this primed zero, go to step 5. Otherwise, cover this row and uncover the column containing the starred zero. Continue this process until there are no uncovered zeros left. Save the smallest uncovered element and go to step 6.

Step 5: Construct a series of alternating primed and starred zeros. Let z_0 represent the uncovered primed zero found in step 4 and z_1 denote the starred zero in the column of z_0 (if any). Continue the procedure until the series terminates at a primed zero that has no starred zero in its column. Unstar each starred zero in the series and replace all primes with stars. Erase all other primes and uncover every line in the cost matrix. Go to step 3.

Step 6: Add the value found in step 4 to every element in covered rows, and subtract it from every element in uncovered columns. Go to step 4 without altering any stars, primes, or covered lines.

References

[1] Y. Chen *et al.*, "Neuromorphic computing's yesterday, today, and tomorrow—An evolutional view," *Integration*, vol. 61, pp. 49–61, Mar. 2018.

[2] D. Gao, D. Reis, X. S. Hu, and C. Zhuo, "Eva-CiM: A system-level performance and energy evaluation framework for computing-in-memory architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 5011–5024, Dec. 2020.

[3] Q. Xu, H. Geng, S. Chen, B. Yu, and F. Wu, "Memristive crossbar mapping for neuromorphic computing systems on 3D IC," *ACM Trans. Design Autom. Electron. Syst.*, vol. 25, no. 1, pp. 1–19, Jan. 2020.

[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. IEEE 6th Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, Aug. 2017, pp. 1–6.

[5] L. Xia *et al.*, "Technological exploration of RRAM crossbar array for matrix-vector multiplication," *J. Comput. Sci. Technol.*, vol. 31, no. 1, pp. 3–19, Jan. 2016.

[6] W. Huangfu *et al.*, "Computation-oriented fault-tolerance schemes for RRAM computing systems," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017, pp. 794–799.

[7] Q. Xu *et al.*, "Fault tolerance in memristive crossbar-based neuromorphic computing systems," *Integration*, vol. 70, pp. 70–79, Jan. 2020.

[8] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[9] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training enabled by on-line fault detection for RRAM-based neural computing systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1611–1624, Sep. 2019.

[10] A. Chaudhuri, B. Yan, Y. Chen, and K. Chakrabarty, "Hardware fault tolerance for binary RRAM crossbars," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019, pp. 1–10.

[11] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Handling stuck-at-fault defects using matrix transformation for robust inference of DNNs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2448–2460, Oct. 2020.

[12] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: Variation-aware training for memristor X-bar," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015, pp. 1–6.

[13] L. Chen *et al.*, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 19–24.

[14] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[15] T. Liu, W. Wen, L. Jiang, Y. Wang, C. Yang, and G. Quan, "A fault-tolerant neural network architecture," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[16] 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. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–4.

[17] A. Chaudhuri and K. Chakrabarty, "Analysis of process variations, defects, and design-induced coupling in memristors," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2018, pp. 1–10.

[18] A. Chaudhuri, M. Liu, and K. Chakrabarty, "Fault-tolerant neuromorphic computing systems," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2019, pp. 1–10.

[19] A. J. Van De Goor and Y. Zorian, "Effective March algorithms for testing single-order addressed memories," *J. Electron. Test.*, vol. 5, no. 4, pp. 337–345, Nov. 1994.

[20] C.-Y. Chen *et al.*, "RRAM defect modeling and failure analysis based on March test and a novel squeeze-search scheme," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 180–190, Jan. 2015.

[21] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu, "Modeling, detection, and diagnosis of faults in multilevel memristor memories," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 34, no. 5, pp. 822–834, May 2015.

[22] A. Chaudhuri, J. Talukdar, F. Su, and K. Chakrabarty, "Functional criticality classification of structural faults in AI accelerators," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–5.

[23] S. Yu, Y. Wu, and H.-S.-P. Wong, "Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory," *Appl. Phys. Lett.*, vol. 98, no. 10, Mar. 2011, Art. no. 103514.

[24] S. R. Lee *et al.*, "Multi-level switching of triple-layered TaOx RRAM with excellent reliability for storage class memory," in *Proc. Symp. VLSI Technol. (VLSIT)*, Jun. 2012, pp. 71–72.

[25] J. L. Bernier, J. Ortega, E. Ros, I. Rojas, and A. Prieto, "A quantitative study of fault tolerance, noise immunity, and generalization ability of MLPs," *Neural Comput.*, vol. 12, no. 12, pp. 2941–2964, 2000.

[26] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 253–256.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[28] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[30] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Int. Conf. Mach. Learn. (ICML)*, 2013, pp. 1058–1066.

[31] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[32] C.-T. Chiu, K. Mehrotra, C. K. Mohan, and S. Ranka, "Modifying training algorithms for improved fault tolerance," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 1. Orlando, FL, USA, Jun./Jul. 1994, pp. 333–338.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[34] H. Zhang, W. Wu, F. Liu, and M. Yao, "Boundedness and convergence of online gradient method with penalty for feedforward neural networks," *IEEE Trans. Neural Netw.*, vol. 20, no. 6, pp. 1050–1054, Jun. 2009.

[35] P. J. Edwards and A. F. Murray, "Weight saliency regularization in augmented networks," in *Proc. Eur. Symp. Artif. Neural Netw. (ESANN)*, 1998, pp. 261–266.

[36] Z. He, T. Zhang, and R. Lee, "Sensitive-sample fingerprinting of deep neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4729–4737.

[37] P. Dey, K. Nag, T. Pal, and N. R. Pal, "Regularizing multilayer perceptron for robustness," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 8, pp. 1255–1266, Aug. 2018.

[38] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 131–138.

[39] G. A. Seber and C. J. Wild, *Nonlinear Regression*. Hoboken, NJ, USA: Wiley, 2003.

[40] J. L. Bernier, J. Ortega, I. Rojas, E. Ros, and A. Prieto, "Obtaining fault tolerant multilayer perceptrons using an explicit regularization," *Neural Process. Lett.*, vol. 12, no. 2, pp. 107–113, Oct. 2000.

[41] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 6391–6401.

[42] Y. Van de Burgt *et al.*, "A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing," *Nature Mater.*, vol. 16, no. 4, pp. 414–418, Apr. 2017.

[43] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 14–26.

[44] Q. Xu, J. Wang, H. Geng, S. Chen, and X. Wen, "Reliability-driven neuromorphic computing systems design," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1586–1591.

[45] F. Bourgeois and J.-C. Lassalle, "An extension of the Munkres algorithm for the assignment problem to rectangular matrices," *Commun. ACM*, vol. 14, no. 12, pp. 802–804, Dec. 1971.

[46] J. K. Wong, "A new implementation of an algorithm for the optimal assignment problem: An improved version of Munkres' algorithm," *BIT*, vol. 19, no. 3, pp. 418–424, Sep. 1979.

[47] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 541–552.

[48] B. K. Joardar, J. R. Doppa, P. P. Pande, H. Li, and K. Chakrabarty, "AccuReD: High accuracy training of CNNs on ReRAM/GPU heterogeneous 3-D architecture," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 5, pp. 971–984, May 2021.

[49] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

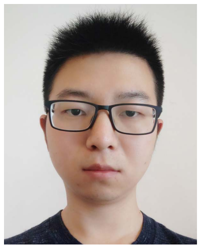
[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[51] A. Krizhevsky, "Learning multiple layers of features from tiny images," Citeseer, Princeton, NJ, USA, Tech. Rep. 7, 2009.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[53] B. Feinberg, S. Wang, and E. Ipek, "Making memristive neural network accelerators reliable," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 52–65.

Qi Xu (Member, IEEE) received the Ph.D. degree in electronic science and technology from the University of Science and Technology of China (USTC), Hefei, China, in 2018. He is currently an Associate Professor with the School of Microelectronics, USTC. His research interests include physical design automation and design for reliability for 3-D integrated circuits.



Junpeng Wang received the B.S. degree in applied physics from the University of Science and Technology of China (USTC), Hefei, China, in 2018, where he is currently pursuing the Ph.D. degree with the School of Microelectronics. His current research interests include hardware acceleration of deep neural networks and processing-in-memory systems.

Bo Yuan (Member, IEEE) received the B.Sc. and Ph.D. degrees in electronic information science and technology from the University of Science and Technology of China (USTC), Hefei, China, in 2009 and 2014, respectively. From May 2012 to April 2013, he was a Visiting Student with the School of Computer Science, University of Birmingham, Birmingham, U.K. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen, China. His current research interests include evolutionary computation, electronic design automation, and machine learning.

Qi Sun (Graduate Student Member, IEEE) received the B.Eng. degree in computer science from Xidian University in 2018. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His current research interests include deep neural network hardware acceleration, high-level synthesis, and design space exploration.

Song Chen (Member, IEEE) received the B.S. degree in computer science from Xi'an Jiaotong University, China, in 2000, and the Ph.D. degree in computer science from Tsinghua University, China, in 2005. He served at the Graduate School of Information, Production and Systems, Waseda University, Japan, as a Research Associate from August 2005 to March 2009, and an Assistant Professor from April 2009 to August 2012. He is currently an Associate Professor with the School of Microelectronics, University of Science and Technology of China (USTC). His research interests include several aspects of VLSI design automation, on-chip communication system, in-memory computing, and computer-aided design for emerging technologies. He is a member of ACM and IEICE.

Bei Yu (Member, IEEE) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He received seven best paper awards from ASPDAC 2021, ICTAI 2019, *Integration* (the VLSI journal) in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, ICCAD 2013, ASPDAC 2012, and six ICCAD/ISPD contest awards. He has served as the TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is an Editor of IEEE TECHNICAL COMMITTEE ON CYBER-PHYSICAL SYSTEMS NEWSLETTER.

Yi Kang (Member, IEEE) received the B.S. and M.S. degrees in electronic engineering from Tsinghua University and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign. He is currently a Professor with the School of Microelectronics, University of Science and Technology of China (USTC). Before went to teaching in USTC, he worked as the Chief Scientist and an SVP with Spreadtrum Communications Inc., Shanghai, China. His current research area includes new computing and memory architecture and implementation of neural networks.

Feng Wu (Fellow, IEEE) received the B.S. degree in electrical engineering from Xidian University in 1992 and the M.S. and Ph.D. degrees in computer science from the Harbin Institute of Technology in 1996 and 1999, respectively. He is currently a Professor and an Assistant to the President with the University of Science and Technology of China (USTC), Hefei, China. Previously, he was a Principle Researcher and the Research Manager with Microsoft Research Asia, Beijing, China. He has authored or coauthored over 120 journal articles (including several IEEE TRANSACTIONS) and top conference papers on MOBICOM, SIGIR, CVPR, and ACMMM. He has 80 granted U.S. patents. His 15 techniques have been adopted into international video coding standards. His research interests include various aspects of video technology and artificial intelligence. He received the best paper awards in IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY in 2009, VCIP in 2007 and 2016, and PCM in 2008, and the Best Associate Editor Award of IEEE TRANSACTIONS ON IMAGE PROCESSING in 2018. He also serves as the General Chair for ICME 2019, and the TPC Chair for MMSP 2011, VCIP 2010, and PCM 2009. He serves or had served as the Deputy Editor-in-Chief for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY and an Associate Editor for IEEE TRANSACTIONS ON IMAGE PROCESSING and IEEE TRANSACTIONS ON MULTIMEDIA.