

# Fault-tolerance Mapping of Spiking Neural Networks to RRAM-based Neuromorphic Hardware

Yuqing Xiong, Cao Xiao, Zhijie Yang, Lei Wang, and Mengying Zhao

**Abstract**—Spiking neural networks (SNNs) have been widely used in artificial intelligence applications. Resistive random-access memory (RRAM) based neuromorphic hardware can be used for low-power and high-speed inference of SNNs. However, RRAM devices suffer from stuck-at-fault (SAF) defects due to the immature fabrication process. SAF defects can lead to incorrect weights of SNNs and thus severely degrade the inference accuracy. In this paper, we propose a fault-tolerance synaptic-to-RRAM mapping scheme to deploy SNNs while protecting the inference accuracy. We first explore how different weights affect the model accuracy in SNNs and find that the frequency of spikes plays a vital role. Motivated by this, we develop an SNN-oriented metric to evaluate the importance of weights. Then we propose a defect-aware mapping scheme based on a simulated-annealing framework to efficiently map synapses to RRAM to mitigate the impact of SAFs. Evaluation shows that the proposed strategy can improve the accuracy by 18.74% on average compared to existing mapping strategy.

## I. INTRODUCTION

Spiking Neural Networks (SNNs) employ spike-based computation and biologically inspired learning mechanisms to mimic the information-processing principles of the human brain [1]. In SNNs, neurons communicate by transmitting spikes to one another through synapses. Computation in SNNs is driven by asynchronously occurring spike trains, where the location and frequency of spikes govern information processing within the network. Leveraging the sparsity of spike and event-driven operations, SNNs running on neuromorphic hardware [2]–[5] show strong potential to achieve high accuracy in classification tasks (e.g., digit classification, object recognition) with significantly lower energy consumption than artificial neural networks (ANNs) [6].

To enhance energy efficiency and throughput, both SNNs and ANNs can be deployed on resistive random-access memory (RRAM) crossbars, where analog-domain computations are enabled by emerging RRAM devices [7]–[9]. RRAM cells support multi-level state representation, where each programmable cell encodes synaptic weights as resistance or conductance. For SNNs, synaptic weights can be mapped to the conductance of RRAM cells, and spikes can be encoded as input voltages. In this way, the RRAM crossbar can efficiently perform multiply-and-accumulate (MAC) operations according to Kirchhoff’s law [10], as shown in Fig. 1 (a). This makes RRAM an ideal hardware device for energy-efficient SNN deployment in edge scenarios [8].

This work was supported by Young Talent of Lifting engineering for Science and Technology in Shandong, China (Grant No. SDAST2024QTA078), and the National Natural Science Foundation of China (Grants Nos. 62372270, 62372461, 62032001, 62203457, and 62406335). Corresponding authors: Mengying Zhao and Zhijie Yang.

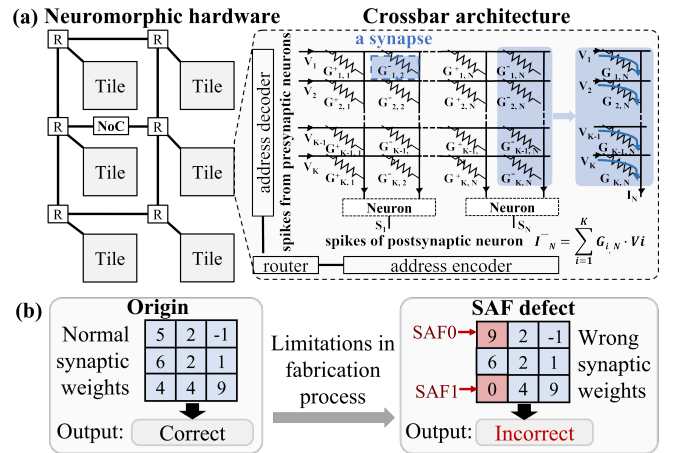


Fig. 1. (a) Architecture of RRAM-based neuromorphic hardware. (b) SAF defects lead to wrong weights, resulting in incorrect output.

However, due to fabrication and forming process limitations [11], static hard failure of stuck-at-fault (SAF) defects can occur in RRAM devices. SAF defects can make RRAM stuck at high/low resistance level, resulting in incorrect outputs, as shown in Fig. 1 (b). The weight values of SNN can be distorted by SAF defects, which may cause over 20% degradation in SNN inference accuracy [12]. To detect the locations of SAF, algorithms such as March-C and squeeze-search are proposed to compare the actual resistance of RRAM cells with reference resistance using multi-level reference voltages [11], [13], [14].

With the identifications of SAFs, fault-tolerance strategies need to be further employed to achieve robust SNN inference. Some related work focuses on handling errors for SNN inference on dynamic random-access memory (DRAM) [15]–[17], which cannot be directly applied to RRAM due to different hardware characteristics. In the context of RRAM crossbars, to mitigate the impact of SAFs, a hardware approach introduces redundant rows to recover computation results, at the cost of higher area and power consumption [18]–[20]. In a software approach, retaining or remapping can be effective. Retraining typically fixes the gradients of faulty weights to zero and updates the remaining parameters through additional rounds of training [21], [22]. Remapping aims to select a mapping scheme that minimizes accuracy degradation by evaluating a large number of candidate mappings [23]–[25].

However, the above schemes are specifically designed for ANNs, but not SNNs. The main difference between ANNs and SNNs lies in that SNNs rely on spikes to activate weights to propagate, so the importance of weights is highly dependent on the frequency of spikes. Consequently, we need to

fully explore the feature of SNNs to propose a defect-aware synapse-to-RRAM mapping for robust inference on RRAMs. Specifically, the main contributions are as follows.

- We explore the feature of spikes in SNN models, and propose a metric with spike frequency to evaluate the importance of weights.
- We develop a heuristic defect-aware fault-tolerance mapping algorithm to efficiently determine the synapse-to-RRAM mapping, protecting important weights from being mapped to defective RRAM to improve accuracy.
- We evaluate the proposed algorithm on SNN models with different scales and compare it with related work to show its accuracy improvement.

## II. BACKGROUND AND RELATED WORK

In this section, we present the background on the SNNs and RRAM, followed by a discussion of related work on fault-tolerance algorithms.

### A. Spiking Neural Networks

A key characteristic of SNNs is that their neurons process inputs and outputs as sequences of spikes with unit amplitude propagated on neuromorphic hardware. Fig. 2 illustrates an example of SNN, where inputs are encoded into spike trains and propagated through synaptic weights. Neurons integrate these spike potentials via synapses and fire binary spikes (0/1) once the membrane voltage exceeds a threshold, thereby producing spike output.

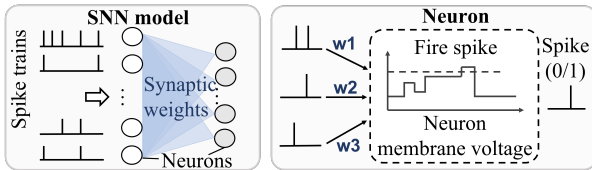


Fig. 2. An example of SNN model.

### B. SAF in RRAM-based accelerator

Fig. 1(b) shows an RRAM-based neuromorphic architecture with multiple computational tiles interconnected via a network-on-chip (NoC) for spike communication [26]. Routers encode spikes into address packets for flexible delivery [27], while each tile implements neurons and synapses using crossbars for spike transmission and weight storage. Due to hardware resource limits, large-scale SNNs must be partitioned into neural clusters [28], mapped to tiles, and then assigned to RRAM crossbar rows/columns. Existing mapping approaches like SpiNeMap [29] minimize spike communication distance but ignore RRAM’s stuck-at-fault (SAF) defects—caused by immature fabrication [11].

SAF defect can be detected through March-C and squeeze-search algorithm [11], [13], [14]. Even a 0.02 SAF rate can degrade inference accuracy by over 20 [12]. SAF0 denotes cells stuck at low-resistance (LRS), e.g., due to overforming; SAF1 denotes cells stuck at high-resistance (HRS), e.g., from broken word lines. Both cause incorrect synaptic weights and accuracy loss.

### C. Related Work

To alleviate SAF defects in RRAM, redundant hardware is employed to enhance robustness by representing weights, but this introduces additional hardware overhead [18]–[20]. Therefore, some software-based methods have been proposed to address SAF defects in ANN. During backpropagation, the retraining method sets the gradients of defective weights to zero for fault tolerance, which may cause the training to fail to converge [21], [22]. The remapping method uses routers to perform row-column permutation to minimize the custom cost function and avoid significant differences between the values defined by SAF defects and the original weights [23]–[25].

However, the evaluation criteria used in ANN-based fault-tolerance algorithms are not well suited for SNNs. In ANNs, large weights are typically deemed critical to model accuracy, and thus prioritized for mapping onto defect-free RRAM cells. SNNs exhibit event-driven feature, where large weights may be activated infrequently due to sparse spike activity. Therefore, both the weight deviation and spike frequency must be jointly considered to precisely evaluate the fault impact in SNNs.

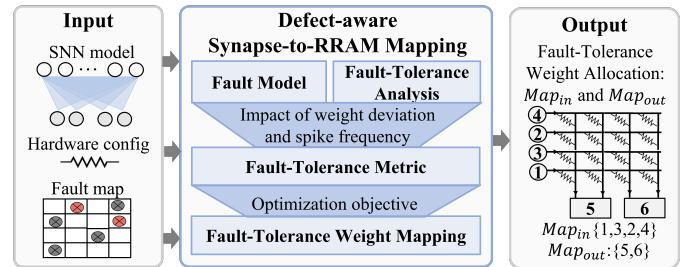


Fig. 3. High-level overview of the proposed defect-aware synapse-to-RRAM mapping.

## III. DEFECT-AWARE SYNAPSE-TO-RRAM MAPPING

The high-level overview of our proposed defect-aware synapse-to-RRAM mapping is shown in Fig. 3. It takes the target SNN model, the hardware configuration including cell resolution, and fault map of RRAM crossbar as inputs. The fault map indicates the exact location of SAF defects<sup>1</sup>. For fault-tolerance mapping, we first integrate the fault information into the SNN model, and then conduct a quantitative fault-tolerance analysis based on the fault model. Next, we design a fault-tolerance metric to evaluate mapping schemes accordingly, which is then used as an optimization objective to be minimized. Finally, a simulated-annealing based fault-tolerance mapping algorithm is proposed to achieve efficient defect-aware synapse-to-RRAM mapping.

### A. Fault Model

The way SAF defects affect weights needs to be modeled. Each RRAM cell in a crossbar stores a weight as conductance  $G$ , mapped from programmable resistance. Weights are encoded into the conductance range  $[G_{\min}, G_{\max}]$  [30]:

<sup>1</sup>Existing techniques such as March-C and squeeze-search can be used for SAF location detection [11], [13], [14].

$$G_w = \frac{|w|}{|W|_{\max}} (G_{\max} - G_{\min}) + G_{\min} \quad (1)$$

where  $|W|_{\max}$  is the maximum absolute weight. Under SAF0, the RRAM cell is stuck at LRS, making  $G_w$  fixed at  $G_{\max}$  and mapping weights to  $\pm|W|_{\max}$ . Under SAF1, it is stuck at HRS, fixing  $G_w = G_{\min}$  and mapping weights to 0. As illustrated in Fig. 1 (b), SAF0 forces weights to the maximum (e.g., 9), while SAF1 forces them to 0. Thus, we can transform the hardware SAF defects into biased weights, which can be used in software simulation for SNN behaviors.

### B. Fault-tolerance Analysis

Before synapse-to-RRAM mapping, it is necessary to analyze weight importance to protect critical ones from SAF defects during mapping. Since SNN computation depends on both weights and spikes, we use a simple two-layer fully connected network on the MNIST as a case study to analyze the effect of weight deviation and spike frequency on accuracy.

#### 1) Impact of weight deviation on accuracy

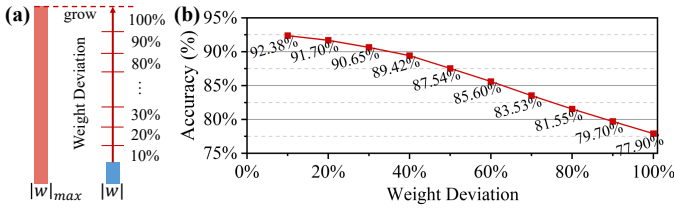


Fig. 4. Impact of weight deviation on accuracy in a two-layer fully connected SNN on the MNIST dataset. (a) Classification of weight deviation levels. (b) Accuracy under different degrees of weight deviation.

We inject controlled deviations into model weights to evaluate their effect on accuracy. As shown in Fig. 4 (a), each weight  $w$  is gradually increased toward  $w_{\max}$  in 10% steps. The results shown in Fig. 4 (b) indicate that as the degree of weight deviation increases, inference accuracy obviously degrades. This suggests that fault-tolerance mapping algorithm should minimize weight deviation to preserve model accuracy.

#### 2) Impact of spike frequency on accuracy

In SNNs, synapses not only store weight values but also embody spike frequency information, which to some extent indicates the importance of weights.

To analyze how spike frequency affects model accuracy, we first group weights based on the spike frequency of the neurons that activate them. In 100 timesteps of spike generation, neurons are sorted in descending order by their spike frequency and then divided evenly into 10 groups. As shown in the spike frequency distribution in Fig. 5 (a), the spike frequency of a large number of neurons is close to zero. The difference in spike frequency between neurons is large. To eliminate the confounding effect of weight values on the analysis of spike frequency, we present the weight distribution corresponding to each spike frequency group in Fig. 5(b). It can be observed that the weights within each group are approximately normally distributed around zero, with little observable bias toward extreme values.

To assess the impact of spike frequency, we inject SAF0 faults by setting affected weights to their maximum value.

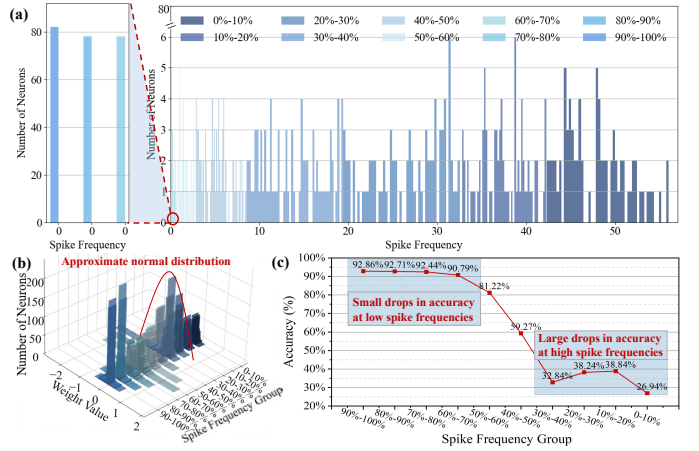


Fig. 5. Impact of spike frequency on accuracy in a two-layer fully connected SNN on the MNIST dataset. (a) Frequency distribution and grouping of neuronal firing spikes. (b) Distribution of weight sizes involved in each group of neurons. (c) Accuracy under faults occurring at different spike frequency groupings.

As shown in Fig. 5 (c), faults on low-frequency weights (60~100%) cause only low accuracy loss (with accuracy dropped to range of 90.79~92.86%), while faults on high-frequency weights (0~30%) lead to severe accuracy loss (with accuracy dropped to range of 26.94~38.84%). Thus, fault-tolerant mapping should prioritize protecting high-frequency weights.

### C. Fault-tolerance Metric

Based on the previous fault-tolerance analysis, we define a fault-tolerance metric to evaluate mapping schemes in a  $n \times n$  crossbar. Let  $x = Map_{in}[i]$  and  $y = Map_{out}[j]$  denote the pre-synaptic neuron  $x$  and post-synaptic neuron  $y$  mapped to row  $i$  and column  $j$ . For each synapse  $w_{x,y}$  mapped to RRAM  $r_{i,j}$ , the error is:

$$ErrorD_{i,x,j,y} = \begin{cases} 0, & \text{if } r_{i,j} = \text{normal} \\ |w_{x,y}| - |W|_{\max} \cdot s_{x,y}, & \text{if } r_{i,j} = \text{SAF0} \\ |w_{x,y}| - 0 \cdot s_{x,y}, & \text{if } r_{i,j} = \text{SAF1} \end{cases} \quad (2)$$

where  $s_{x,y}$  is the spike frequency of the synapse  $w_{x,y}$  in a given SNN workload. Normal cells yield zero error, while defective ones introduce deviations scaled by spike frequency. The calculations here focuses on SAF defects, without considering other non-ideal factors such as device variability or noise. The optimization seeks mappings  $Map_{in}$ ,  $Map_{out}$  that minimize the total error:

$$\min \sum_{1 \leq i,j \leq n} ErrorD_{i,x,j,y}, \text{ where } x = Map_{in}[i], y = Map_{out}[j] \quad (3)$$

Fig. 6 (a) illustrates how our metric works. The SNN model has three neurons, with synapses carrying both weights and spike frequencies. In Mapping 1, the important synapse A (large weight, high frequency) is placed on SAF1, yielding ErrorD=48, while synapse B (small weight, low frequency) on SAF0 gives ErrorD=12, for a total of 60. In Mapping 2, A is mapped to a normal cell (ErrorD=0) and B to SAF1 (ErrorD=4), reducing the total ErrorD to 4. Thus, ErrorD effectively distinguishes mapping quality and guides better mapping.

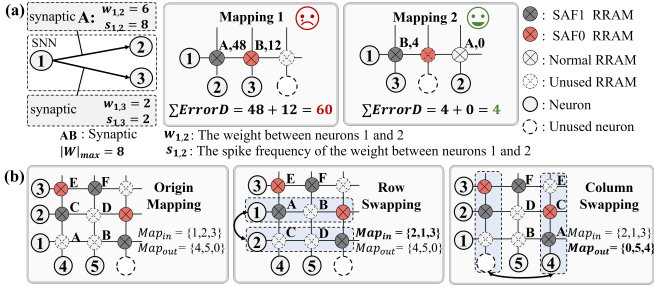


Fig. 6. (a) Example of how ErrorD works. ErrorD reduces mapping costs by considering both weight deviation and spike frequency. (b) Example of how fault-tolerance mapping works. Obtain a new mapping based on random row/column swapping.

### Algorithm 1 Defect-aware Synapse-to-RRAM Mapping

**Input:** SNN\_model, HW\_config, fault\_map,  $\gamma$ ,  $T$   
**Output:**  $BestMap_{in}$ ,  $BestMap_{out}$

- 1: Initialize  $Map_{in}$  and  $Map_{out}$  according to SNN\_model and HW\_config.  
*// init mapping*
- 2: **for**  $i = 1$  to  $n$  **do** *// init ErrorD*
- 3:    $Cost = Cost + Cal\_ErrorD(fault\_map, row, i)$ ;
- 4: **end for**
- 5:  $BestCost = Cost$ ;
- 6: **while** Not Exit( $Cost, T$ ) **do**
- 7:    $sel = rand(0,1)$ ;  $\Delta Cost = 0$ ;
- 8:   **if**  $sel < 0.5$  **then** *// Adjust the row*
- 9:     Randomly select  $r_1, r_2$  from  $1 - n$ ;
- 10:      $\Delta Cost - = Cal\_ErrorD(fault\_map, row, r_1, r_2)$ ;
- 11:     Swap  $Map_{in}[r_1]$  and  $Map_{in}[r_2]$ ;
- 12:      $\Delta Cost + = Cal\_ErrorD(fault\_map, row, r_1, r_2)$ ;
- 13:   **else** *// Adjust the column*
- 14:     *// Here is the same as the row operation above*
- 15:   **end if**
- 16:    $p \leftarrow \exp(-\Delta Cost/T)$ ;  $p_{acc} \leftarrow rand(0,1)$ ;
- 17:   **if**  $\Delta Cost < 0$  or  $p > p_{acc}$  **then**
- 18:      $Cost + = \Delta Cost$ ;
- 19:     **if**  $Cost < BestCost$  **then**
- 20:        $BestMap_{in} = Map_{in}$ ;
- 21:        $BestMap_{out} = Map_{out}$ ;
- 22:        $BestCost = Cost$ ;
- 23:     **end if**
- 24:   **else**
- 25:     **if**  $sel < 0.5$  **then** *// Adjust the row*
- 26:       Swap back  $Map_{in}[r_1]$  and  $Map_{in}[r_2]$ ;
- 27:     **else** *// Adjust the column*
- 28:       Swap back  $Map_{out}[c_1]$  and  $Map_{out}[c_2]$ ;
- 29:     **end if**
- 30:   **end if**
- 31:    $T = \gamma \times T$  ;
- 32: **end while**

### D. Fault-tolerance Mapping Algorithm

To improve the fault-tolerance of SNNs on RRAM crossbars, we formulate the synapse-to-RRAM mapping as the minimization problem in Eq.(3), aiming to find the optimal  $Map_{in}$  and  $Map_{out}$ . To avoid the high complexity of enumerating all  $Map_{in/out}$  solution space (i.e.  $O((n!)^2)$ ), candidate mappings can be generated by randomly swapping rows or columns, as illustrated in Fig. 6 (b). Simulated annealing (SA) [31], described in Alg.1, is employed to efficiently search the solution space.

The Alg.1 first initializes the mapping and computes the corresponding cost via  $Cal\_ErrorD$  (Line 1). It then iteratively swaps random rows or columns of the crossbar and evaluates the change in cost,  $\Delta Cost$  (Lines 6–32). Solutions with lower

cost are always accepted, while higher-cost ones may be accepted with a probability determined by the temperature  $T$  (Lines 16-17), which gradually decreases by factor  $\gamma$  (Line 31). This acceptance strategy avoids local optima at high  $T$  and progressively enforces stricter cost reduction as  $T$  cools down. The process terminates when  $T$  falls below a threshold and the cost stabilizes, yielding a low-ErrorD mapping scheme.

## IV. EVALUATION

In this section, we report evaluation results and give discussions.

### A. Experimental Setup

We compare the proposed defect-aware synapse-to-RRAM (DASR) mapping with SpiNeMap [29]. SpiNeMap is a widely used algorithm for mapping SNN onto RRAM-based crossbar [32]–[34], where crossbar rows/columns are assigned sequentially according to neuron indices. Both algorithms integrate with SpikingJelly [35] to conduct SNN inference and fault injection.

TABLE I  
BENCHMARK INFORMATION [35]

Benchmarks	#Neurons	#Synapses	Accuracy <sup>6</sup>
MNIST <sup>1</sup>	$7.94 \times 10^2$	$7.84 \times 10^3$	92.86%
NMNIST <sup>2</sup>	$4.60 \times 10^4$	$2.84 \times 10^6$	98.15%
FashionMNIST <sup>3</sup>	$4.06 \times 10^4$	$2.89 \times 10^6$	93.47%
CIFAR10 <sup>4</sup>	$1.03 \times 10^5$	$1.61 \times 10^7$	83.24%
DVS128Gesture <sup>5</sup>	$9.06 \times 10^5$	$5.99 \times 10^7$	77.43%

<sup>1</sup> Input(1\*28\*28)-FC(10)

<sup>2</sup> Input(2\*34\*34)-[Conv.Pool]\*24-[Conv.Pool]\*24-FC(512)-FC(100)-FC(10)

<sup>3</sup> Input(1\*28\*28)-[Conv.Pool]\*32-[Conv.Pool]\*32-FC(512)-FC(100)-FC(10)

<sup>4</sup> Input(3\*32\*32)-Conv\*24-Conv\*24-[Conv.Pool]\*24-Conv\*24-Conv\*24-[Conv.Pool]\*24-FC(512)-FC(100)-FC(10)

<sup>5</sup> Input(2\*128\*128)-[Conv.Pool]\*32-[Conv.Pool]\*32-[Conv.Pool]\*32-[Conv.Pool]\*32-FC(512)-FC(110)-FC(11)

<sup>6</sup> Model accuracy without defects

Table I lists the five SNN applications used for accuracy evaluation. These applications complete classification tasks on traditional static MNIST [36], CIFAR-10 [37], and FashionMNIST [38] datasets, as well as neuromorphic N-MNIST [39] and DVS128 Gesture [40] datasets. Neurons use the LIF model [41] with 7-bit quantized weights [42].

The hardware platform for the experiment contains 4 crossbars with 256 neurons and input axons in each crossbar based on the general configuration of [43]. We use OxRRAM-based 1T1R as synaptic technology and 45nm node CMOS as neuron technology [33]. The HRS of RRAM device is  $100k\Omega$ , and the LRS is  $3k\Omega$ . Prior work [11], [22] shows that SAF defects usually follow uniform distribution and the ratio of SAF0 to SAF1 is approximately 1:4. So we use the probability configuration of SAF0 and SAF1 to be 0.1% and 0.4% [12] respectively in the following experiments. The probability of defects is also scaled by a factor of two to further evaluate the accuracy recovery capability of DASR at different rates.

### B. Results

#### 1) Comparison on accuracy with SpiNeMap [29]

DASR is primarily designed to improve inference accuracy under SAF defects. Therefore, Table II shows the fault-

tolerance performance of the DASR for all benchmarks. On average, DASR improves accuracy by 18.74% compared to SpiNeMap. For the MNIST dataset, a simple fully connected two-layer network is used. Due to the simplicity of the task and the network, MNIST is insensitive to SAFs as the error weights cannot propagate through the network, resulting in limited accuracy degradation. For this workload, DASR achieves a maximum recovery of 5.98%. In contrast, other models with more complexity exhibit much weaker fault tolerance. For example, for CIFAR-10, which has a more complex network, the error weights introduced by SAFs can affect more neurons in the network, resulting in larger accuracy degradation. However, the DASR algorithm can recover a maximum accuracy of 55.57% in contrast.

TABLE II  
ACCURACY COMPARED WITH SPINeMAP [29]

Benchmarks	Defect Ratio	SpiNeMap [29] Accuracy	DASR Accuracy	Improved Accuracy
MNIST	0.25%	92.67%	92.84%	0.17%
	0.50%	90.58%	92.85%	2.27%
	1.00%	86.83%	92.81%	5.98%
NMNIST	0.25%	95.52%	97.66%	2.14%
	0.50%	75.55%	96.56%	21.01%
	1.00%	38.09%	93.59%	55.50%
Fashion MNIST	0.25%	84.22%	92.97%	8.75%
	0.50%	82.60%	92.45%	9.85%
	1.00%	60.83%	82.70%	21.87%
CIFAR10	0.25%	63.28%	85.16%	21.88%
	0.50%	45.31%	78.52%	33.21%
	1.00%	28.91%	84.48%	55.57%
DVS128 Gesture	0.25%	57.29%	75.37%	18.08%
	0.50%	61.81%	73.05%	11.24%
	1.00%	49.31%	62.95%	13.64%
Average				18.74%

### 2) Comparison on energy with SpiNeMap [29]

Hardware energy consumption consists of computation and communication. Since DASR does not use an additional crossbar, the number of occupied crossbars remains the same, the computational energy remains nearly unchanged. Therefore, the comparison focuses on communication energy, which is also a bottleneck in large-scale neuromorphic hardware [29]. As shown in Fig. 7, the communication energy of DASR is 8.22% lower than that of SpiNeMap on average. This experiment adopts the communication energy evaluation method and configuration from NeuroXplorer [33]. As the defect rate increases, more unexpected spikes are generated, which increases the burden on communication between crossbars. DASR reduces the generation of unexpected spikes by decreasing the deviation of important weights, resulting in a decrease in communication energy.

### 3) Insights within simulated annealing

The DASR methodology is initialized with a temperature of 100 and a discount factor of 0.995 in the SA process. Annealing ends when the cost remains unchanged for 400 iterations. Fig. 8 illustrates the temperature and ErrorD evolution during

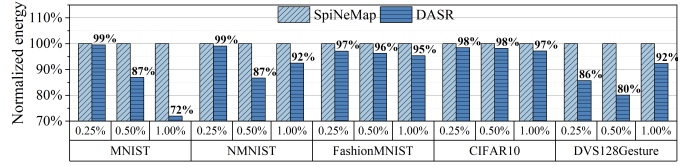


Fig. 7. Comparison of communication energy with SpiNeMap [29].

synapse-to-RRAM mapping for MNIST. At the beginning, the high temperature allows a greater probability of accepting swaps, causing a brief increase in ErrorD to 151.21. As iterations proceed, DASR increasingly favors swaps with lower ErrorD. Both temperature and ErrorD gradually decrease, and fluctuations narrow to 0.81~2.12, leading the mapping scheme toward convergence.

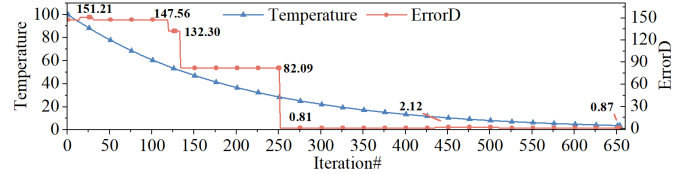


Fig. 8. Change of temperature and ErrorD in DASR.

## C. Discussions

### 1) With either only SAF0 or SAF1 defects

To evaluate fault-tolerance under a single defect type, Fig. 9 illustrates the accuracy recovery results at different defect probabilities using MNIST as an example. Fig. 9 shows that SAF0 has a stronger negative impact than SAF1. When 0.8% of RRAM cells are SAF0 defective, model accuracy drops to 66.90%. In contrast, when 10% of RRAM cells are SAF1 defective, the accuracy is still 91.18%. This occurs because SAF0 generates higher currents on the bitline, leading to increased neuronal spike rates, while SAF1 reduces spike rates, with limited effect if the synapse weight is low or receives few spikes. These findings indicate that SAF0 defects can cause severer accuracy degradation compared to SAF1 defects.

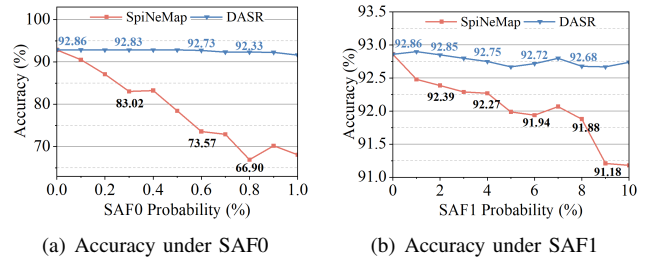


Fig. 9. Accuracy compared to SpiNeMap [29].

### 2) Comparison with ANN fault-tolerance metric [23]

To evaluate the impact of spike frequency in the fault-tolerance metric, we compare DASR with DASR-ANN, a variant that only considers weight deviation. Taking the MNIST application as an example, Fig. 10 shows that DASR-ANN exhibits inferior and less stable fault recovery performance compared to DASR. On average, DASR improves the accuracy by 7.39% and 0.23% compared with DASR-ANN in SAF0 and SAF1, respectively. For the SAF0 probability of 0.6%, the

accuracy of DASR-ANN only reaches 73.41%, which shows an unstable fault-tolerance performance. This highlights the limitation of considering only large weight deviations without accounting for spike frequency.

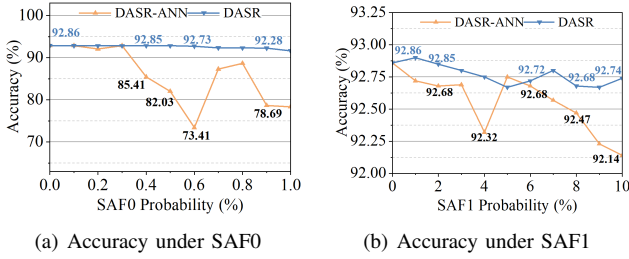


Fig. 10. Accuracy compared to ANN fault-tolerance metric [23].

### 3) Evaluation with various scale of crossbar

We evaluate the robustness of DASR based on the number and size of crossbars. Fig. 11 evaluates DASR’s accuracy recovery on MNIST with 4, 8, 16, 32, and 64 crossbars. SpiNeMap achieves accuracy of 80.13~92.71%, while DASR improves it to 90.71~92.91% with an average improvement of 3.91%, demonstrating stable fault-tolerance across different numbers of crossbar. To assess performance on large-scale crossbars, Fig. 12 shows results for crossbar sizes of 64, 128, 256, 512, and 1024 with 4 crossbars. As the size increases, SpiNeMap’s average accuracy drops from 92.63% to 86.05%, whereas DASR maintains 89.91~92.86%, with an average improvement of 2.95%, indicating superior fault tolerance across varying crossbar sizes.

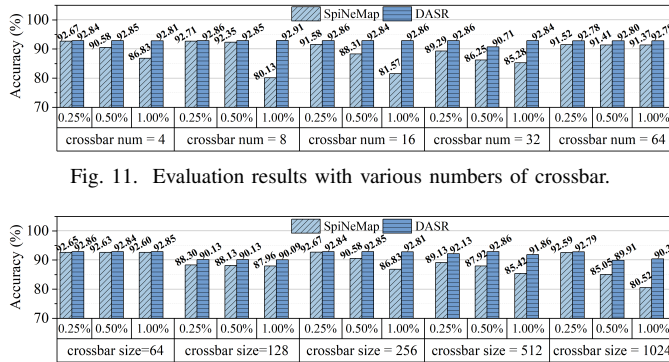


Fig. 12. Evaluation results with various sizes of crossbar.

### 4) Overhead analysis

Any fault-tolerant algorithm, including DASR, requires a fault map indicating the type and location of SAFs, which can be obtained using March-C or squeeze-search [11], [13], [14]. Spike communication is supported by the router in [27], where the content addressable memory (CAM) must be configured to map the SNN model. The DASR mapping scheme can be implemented by updating the CAM. The runtime analysis of setting up these mapping schemes is presented below.

Table III reports the runtime of DASR and KM algorithm [23] at a defect rate of 0.5%. The time measurement is performed with an Intel Platinum 8480+ processor. DASR mainly spends time in the SA phase to find synaptic mapping schemes, so the runtime increases approximately linearly with the number of synapses. A traditional way to implement

TABLE III  
REQUIRED RUNTIME FOR DASR AND KM ALGORITHM [23]

Benchmark	MNIST	NMNIST	Fashion MNIST	CIFAR10	DVS128 Gesture
DASR	4sec	14min	21min	1h 36min	7h 7min
KM [23]	1min	5h 30min	5h 57min	10h+	10h+

synapse-to-RRAM mapping is to use the KM algorithm to find the minimum-cost bipartite matching of the metric. The KM algorithm constructs a cost matrix with cubic complexity, leading to very long runtimes on large models such as CIFAR10 and DVS128Gesture (over 10 hours). In the MNIST, NMNIST and Fashion datasets with existing results, the average improvement of KM algorithm and DASR algorithm is 11.96% and 11.04%, respectively. DASR adopts a heuristic algorithm that only requires exploration of partial mapping schemes on the initial mapping, which is 19.62 times faster than KM algorithm on average, at the cost of only a 0.92% accuracy loss. This shows the efficiency of the DASR algorithm in runtime.

Fig 13 shows the trade-off between fault-tolerance and runtime of DASR in the case of MNIST. The discount factor  $\gamma$  and the termination threshold are combined as termination conditions for the SA algorithm. The termination threshold is defined as the number of stable rounds with unchanged cost. A smaller  $\gamma$  improves accuracy through deeper exploration but increases runtime, while  $\gamma > 0.995$  may accept poor solutions and fall into local optima, reducing fault tolerance. Larger termination thresholds enhance tolerance by forcing more swaps but prolong runtime. Our experiment takes an intermediate setting of  $\gamma=0.995$  and threshold=400 as a balanced solution.

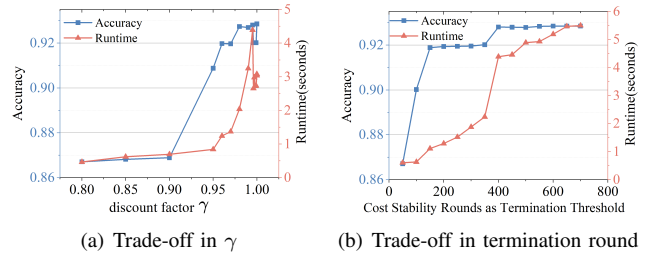


Fig. 13. Trade-off analysis between performance and runtime.

## V. CONCLUSION

In this paper, we propose DASR, a method to mitigate SAF defects in RRAM-based neuromorphic hardware. By analyzing weights and spike frequency, DASR defines a fault-tolerance metric and applies defect-aware synapse-to-RRAM mapping to minimize the induced error during the mapping process. In this way, weights with high spike frequency are avoided from being mapped onto defective RRAM and thus it reduces the deviation between actual and ideal weights. Evaluation shows DASR improves accuracy by 18.74% on average compared to existing mapping strategy.

## REFERENCES

- [1] W. Maass, “Networks of spiking neurons: The third generation of neural network models,” *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997.
- [2] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, “The SpiNNaker Project,” *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

- [3] M. V. DeBole, B. Taba, A. Amir, F. Akopyan, A. Andreopoulos, W. P. Risk, J. Kusnitz, C. Ortega Otero, T. K. Nayak, R. Appuswamy, P. J. Carlson, A. S. Cassidy, and P. Datta, "TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years," *Computer*, vol. 52, no. 5, pp. 20–29, 2019.
- [4] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, and C.-K. Lin, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
- [5] Z. Yang, L. Wang, Y. Wang, L. Peng, X. Chen, X. Xiao, Y. Wang, and W. Xu, "Unicorn: a multicore neuromorphic processor with flexible fan-in and unstrained fan-out for neurons," in *DAC*, 2022, p. 943–948.
- [6] R. V. W. Putra and M. A. Shafique, "FSpiNN: An Optimization Framework for Memory-Efficient and Energy-Efficient Spiking Neural Networks," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3601–3613, 2020.
- [7] S. Park, A. Sheri, J. Kim, J. Noh, J. Jang, M. Jeon, B. Lee, B. R. Lee, B. H. Lee, and H. Hwang, "Neuromorphic speech systems using advanced ReRAM-based synapse," in *IEDM*, 2013, pp. 25.6.1–25.6.4.
- [8] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, and S. S. and, "Neuromorphic computing using non-volatile memory," pp. 89–124, 2017.
- [9] A. Mallik, D. Garbin, A. Fantini, D. Rodopoulos, R. Degraeve, J. Stuijt, A. K. Das, S. Schaafsma, P. Debacker, G. Donadio, H. Hody, L. Goux, G. S. Kar, A. Furnemont, A. Mocuta, and P. Raghavan, "Design-technology co-optimization for OxRRAM-based synaptic processing unit," in *VLSIT*, 2017, pp. T178–T179.
- [10] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *DAC*, 2016, pp. 1–6.
- [11] 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," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 180–190, 2015.
- [12] M. K. F. Lee, Y. Cui, T. Somu, T. Luo, J. Zhou, W. T. Tang, W.-F. Wong, and R. S. M. Goh, "A System-Level Simulator for RRAM-Based Neuromorphic Computing Chips," *ACM Trans. Archit. Code Op.*, vol. 15, no. 4, Jan 2019.
- [13] A. van de Goor and Y. Zorian, "Effective march algorithms for testing single-order addressed memories," in *EDAC*, 1993, pp. 499–505.
- [14] Y.-X. Chen and J.-F. Li, "Fault modeling and testing of 1T1R memristor memories," in *VTS*, 2015, pp. 1–6.
- [15] R. V. W. Putra, M. A. Hanif, and M. Shafique, "EnforceSNN: Enabling resilient and energy-efficient spiking neural network inference considering approximate DRAMs for embedded systems," *Front. Neurosci.*, vol. 16, 2022.
- [16] R. V. W. Putra, M. A. Hanif, and M. Shafique, "SoftSNN: low-cost fault tolerance for spiking neural network accelerators under soft errors," in *DAC*, 2022, p. 151–156.
- [17] R. V. W. Putra, M. A. Hanif, and M. Shafique, "RescueSNN: enabling reliable executions on spiking neural network accelerators under permanent faults," *Front. Neurosci.*, vol. 17, 2023.
- [18] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier, "Multicore soft error rate stabilization using adaptive dual modular redundancy," in *DATE*, 2010, pp. 27–32.
- [19] R. E. Lyons and W. Vanderkulk, "The Use of Triple-Modular Redundancy to Improve Computer Reliability," *IBM Journal of Res. Dev.*, vol. 6, no. 2, pp. 200–209, 1962.
- [20] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at Fault Tolerance in RRAM Computing Systems," *IEEE J. Emerg. Sel. Top Circuits Syst.*, vol. 8, no. 1, pp. 102–115, 2018.
- [21] 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 *DAC*, 2017, pp. 1–6.
- [22] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *DAC*, 2017, pp. 1–6.
- [23] 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 *DATE*, 2017, pp. 19–24.
- [24] Y. Ma, L. Zheng, and P. Zhou, "A Mapping Method Tolerating SAF and Variation for Memristor Crossbar Array Based Neural Network Inference on Edge Devices," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 2, May 2023.
- [25] 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 Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2448–2460, 2020.
- [26] A. Balaji, Y. Wu, A. Das, F. Catthoor, and S. Schaafsma, "Exploration of Segmented Bus As Scalable Global Interconnect for Neuromorphic Computing," in *GLSVLSI*, 2019, p. 495–499.
- [27] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)," *IEEE Trans. Biomed. Circuits and Syst.*, vol. 12, no. 1, pp. 106–122, 2018.
- [28] O. Jin, Q. Xing, Y. Li, S. Deng, S. He, and G. Pan, "Mapping Very Large Scale Spiking Neuron Network to Neuromorphic Hardware," in *ASPLOS*, 2023, p. 419–432.
- [29] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping Spiking Neural Networks to Neuromorphic Hardware," *VLSI*, vol. 28, no. 1, pp. 76–86, 2020.
- [30] F. Hu, Z. You, P. Liu, and J. Kuang, "Circuit Design of Convolutional Neural Network Based on Memristor Crossbar Arrays," *Journal of Computer Research and Development*, vol. 55, no. 05, pp. 1097–1107, 2018.
- [31] P. J. M. Laarhoven and E. H. L. Aarts, *Simulated annealing: theory and applications*. USA: Kluwer Academic Publishers, 1987.
- [32] T. Titirsha, S. Song, A. Das, J. Krichmar, N. Dutt, N. Kandasamy, and F. Catthoor, "Endurance-Aware Mapping of Spiking Neural Networks to Neuromorphic Hardware," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 2, pp. 288–301, 2022.
- [33] A. Balaji, S. Song, T. Titirsha, A. Das, J. Krichmar, N. Dutt, J. Shackelford, N. Kandasamy, and F. Catthoor, "NeuroXplorer 1.0: An Extensible Framework for Architectural Exploration with Spiking Neural Networks," ser. ICONS, 2021.
- [34] S. Song, T. Titirsha, and A. Das, "Improving Inference Lifetime of Neuromorphic Systems via Intelligent Synapse Mapping," in *ASAP*, 2021, pp. 17–24.
- [35] W. Fang, Y. Chen, J. Ding, Z. Yu, T. Masquelier, D. Chen, L. Huang, H. Zhou, G. Li, and Y. Tian, "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence," *Sci. Adv.*, vol. 9, no. 40, p. eadi1480, 2023.
- [36] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Proc. Mag.*, vol. 29, no. 6, pp. 141–142, 2012.
- [37] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," *University of Toronto*, 05 2009.
- [38] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.
- [39] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades," *Front. Neurosci.*, vol. 9, 2015.
- [40] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A Low Power, Fully Event-Based Gesture Recognition System," in *CVPR*, 2017, pp. 7388–7397.
- [41] E. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, 2003.
- [42] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, Jan 2012.
- [43] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramanian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016, pp. 14–26.