Fault-Tolerant Training Enabled by On-Line Fault Detection for RRAM-Based Neural Computing Systems

Lixue Xia, Student Member, IEEE, Mengyun Liu, Xuefei Ning, Krishnendu Chakrabarty, Fellow, IEEE, Yu Wang, Senior Member, IEEE

Abstract—An RRAM-based computing system (RCS) is an attractive hardware platform for implementing neural computing algorithms. On-line training for RCS enables hardware-based learning for a given application and reduces the additional error caused by device parameter variations. However, a high occurrence rate of hard faults due to immature fabrication processes and limited write endurance restrict the applicability of on-line training for RCS. We propose a fault-tolerant on-line training method that alternates between a fault-detection phase and a fault-tolerant training phase. In the fault-detection phase, a quiescent-voltage comparison method is utilized. In the training phase, a threshold-training method and a re-mapping scheme is proposed. Our results show that, compared to neural computing without fault tolerance, the recognition accuracy for the Cifar-10 dataset improves from 37% to 83% when using low-endurance RRAM cells, and from 63% to 76% when using RRAM cells with high endurance but a high percentage of initial faults.

I. INTRODUCTION

Machine learning is now widely used in a variety of domains, and brain-inspired neural computing is considered to be one of the most powerful applications of machine learning. To ensure that neural network algorithms are feasible in practice, hardware implementations require high computing capability and energy efficiency. However, CMOS technology is faced with the bottlenecks of scaling limitations and the “memory wall” for von Neumann architectures [1]. Consequently, the energy efficiency gap between application requirements and hardware implementation continues to grow.

Emerging devices such as metal-oxide resistive random-access memory (RRAM) and its associated crossbar array structure provide the basis for a promising architecture for brain-inspired circuits and systems. By exploiting the crossbar structure, an RRAM-based computing system (RCS) can realize vector-matrix multiplication in analog form and reduce the computational complexity from $O(n^2)$ to $O(1)$. Moreover, since RRAM provides memory for data storage, RCS provides an attractive solution for computing-in-memory. This architecture eliminates the high data transportation overhead that is inherent to von Neumann architectures; therefore, it significantly boosts energy efficiency, particularly for neural computing applications [2], [3].

RRAM faults may occur during both chip fabrication and data processing. RRAM faults can be classified as soft faults and hard faults [4]. For soft faults, the resistance of the RRAM cell can still be tuned, but the actual resistance is different from the expected value. For hard faults, the resistance of an RRAM cell cannot be changed. One approach to tolerate RRAM faults in an RCS is to use an on-line training scheme, which trains a neural network using the output of the RCS [5], [6]. This method can tolerate soft faults by utilizing the inherent fault tolerance capability in neural computing algorithms. However, hard faults still limit the performance of on-line training. The write endurance of a multi-level RRAM cell ranges from $10^6$ to $10^8$ write operations [7], [8], whereas the training phases of neural algorithms typically take $10^5$ to $10^7$ iterations [9], [10]. Since we need to train another network in order to change the neural computing applications for RCS, most cells will become faulty after repeated write operations.

Researchers have shown that more than 50% of the weights in neural networks can be fixed at zero [11], which motivates us to further use neural algorithms to tolerate not only soft faults but also hard faults on RRAM. Two challenges arise in fault-tolerant on-line training in the presence of hard faults. First, we need to know the distribution of faults in RRAM cells during training. The test time of traditional test methods increases quadratically with the number of rows (columns) of the RRAM crossbar [12], and this is a key limiter for on-line testing. Second, traditional redundancy-based methods for memory design cannot be utilized to target hard faults in an RCS. This is because the basic unit of an RCS is an entire RRAM column rather than a single RRAM cell, and redundant columns may also contain, as well as, give rise to hard faults.

In this paper, we present a complete design flow that includes both on-line fault detection and fault-tolerant training for RCS. The main contributions of this paper are as follows:

1) We propose a fault-tolerant training flow with on-line fault detection for an RRAM-based neural computing system. In this flow, the system can periodically detect the current distribution of faults and tolerate the faults in
the next training phase using the inherent fault-tolerant capability of the neural network.

2) We propose an efficient on-line fault detection method by using quiescent-voltage comparisons. Modulo operations are utilized to reduce the hardware overhead. The proposed method can detect faults with more than 85% accuracy and more than 90% fault coverage within an acceptable test time.

3) Approximate performance models are proposed to analyze and optimize the fault detection method. The models are verified by simulation results with relative errors less than 1%.

4) We propose a threshold-training method and a re-mapping scheme during the training phase to tolerate faults using the sparsity of neural networks. Our results show that the accuracy can be restored to the original training accuracy for a fault-free scenario.

II. PRELIMINARIES AND RELATED WORK

A. RRAM-based Neural Computing

An RRAM cell is a passive two-port element with variable resistance, and multiple cells are used to construct a crossbar structure. Researchers have found that if we store a “matrix” on the conductances of RRAM cells in the crossbar structure and input a “vector” as input voltage signals, the RRAM crossbar is able to perform high-efficiency matrix-vector multiplication in analog mode, as shown in Fig. 1(b). Specifically, the relationship between the input and output voltages can be expressed as: \( \tilde{v}_{\text{out},k} = \sum_{j=1}^{N} g_{k,j} \cdot \tilde{v}_{\text{in},j} \), where \( \tilde{v}_{\text{in}} \) is the input voltage vector (denoted by \( j = 1, 2, ..., N \)), \( \tilde{v}_{\text{out}} \) is the output current vector (denoted by \( k = 1, 2, ..., M \)), and \( g_{k,j} \) is the conductance matrix of the RRAM cells representing the matrix data [13]. Based on this structure, several RCS designs have been proposed, e.g., RRAM-based matrix operator [14], [15], RRAM-based neural networks with fully-connected (FC) layers [13], [16], and RRAM-based convolutional neural networks (CNN) with both convolutional (Conv) layers and FC layers [3], [17]. Researchers have successfully demonstrated RRAM-based neural computing with fabricated chips [18], [19].

B. RRAM-based On-line Training

To tolerate soft faults, researchers have demonstrated online training schemes for RRAM and other emerging non-volatile memory (NVM)-based neural networks. As shown in Fig. 1(a), a training iteration contains two steps [11]: 1) a forward-propagation step to obtain classification results using the current network, and 2) a back-propagation step to adjust the weights. Besides the computation of forward-propagation, the back-propagation step also includes matrix-vector multiplication, as shown in (1):

\[
\delta y_i^{n-1}(t) = f'(y_i^{n-1}(t)) \sum_j w_{i,j}^n(t) \delta y_j^n(t)
\]

where \( w_{i,j}^n(t) \) is the weight in the \( i \)th row and \( j \)th column of the \( n \)th layer, \( t \) is the iteration count, \( \delta y_j^n(t) \) is the error signal propagated from the next layer, and \( f'(y_i^n(t)) \) is the derivative of the output in the forward-propagation step.

Since \( \sum_j w_{i,j}^n(t) \delta y_j^n(t) \) is also a matrix-vector multiplication, on-line training structures for RRAM-based neural computing system have been proposed. Yu et al. demonstrated on-line training of RRAM-based binary neural networks (BNN) on a 16 Mb binary RRAM chip [20]. Since this work focuses on NN with binary weights, the number of write operations in training this NN is much smaller than that in training a multi-bit-weight NN. Therefore, the endurance limit is not so severe when the weights of the BNN are updated. This work demonstrates the potential of training NNs on RRAM, but using binary weights will cause accuracy loss in state-of-the-art applications [21]. For multi-bit NNs, Strukov et al. implemented a 9×3 single-layer RRAM-based network through on-line training [5]. This work shows the benefit of on-line training by solving the mapping-variation problem. However, it cannot target hard faults and the endurance problem due to the small size of the array and the simplicity of the application with only a few write operations. Song et al. proposed a pipelined architecture for RRAM-based on-line training [6], and Cheng et al. integrated the training circuits into RRAM memory [22]. These architectures have explored the potential of RRAM-based on-line training to overcome the memory wall, obtaining 10 times more energy efficiency compared to traditional GPU solutions. However, the influence of endurance and hard faults were not considered.

The motivational example in Section III shows the impact of hard faults on the training accuracy and highlights the fact that the network cannot converge in an RCS.

C. Hard Faults in RRAMs

If an RRAM cell contains a hard fault, the resistance of an RRAM cell cannot be changed. This may occur during both chip fabrication and data processing; hard faults include stuck-at-0 (SA0) and stuck-at-1 (SA1) faults caused by
fabrication techniques [23] and limited endurance [8], [24], [25]. The stuck-at-1 (stuck-at low resistance state) faults can be attributed to the overforming problem and short defects during fabrication, and the stuck-at-0 (stuck-at high resistance state) faults are caused by broken wordline and open switch defects [23], [26]. Although the RRAM cell is inherently analog, the faulty cells caused by these mechanisms usually get stuck at the highest and lowest value, i.e., SA0 or SA1 [27], [28]. For hard faults caused by limited endurance, the testing results from fabricated analog RRAM arrays are missing in the literature. However, previous results on binary RRAM cells have shown that a faulty cell usually gets stuck at a state with a high resistance, which is much higher than the highest state of a functional RRAM cell [24]. Based on these results, in this work, we extend the binary fault models to analog RRAM devices.

D. Related Prior Work

1) Fault Detection for RRAM: For on-line fault-tolerant training on a larger network, we also need to know the distribution of faults. Recent work on fault distribution in the RCS are mainly focused on off-line fault detections [12], [29]. These designs can cover many RRAM fault types, and provide high fault coverage. However, the high time complexity limits their usefulness for on-line testing. Moreover, [29] does not support fault localization, and [12] requires sequential March diagnosis to determine fault locations. In [30], a parallel test method was proposed to reduce test time, but this design is not scalable for large crossbars.

2) Fault Tolerance in RRAM-based Computing Systems: Researchers have proposed methods to tolerate hard faults of RCS in the computing phase by redundant schemes [28] and re-training methods [27], [31]; however, these methods cannot tolerate the hard faults in the training phase. Prior work has proposed a framework for fault-tolerant on-line training. Compared to prior work, we propose performance estimation models for the proposed on-line fault-detection method. Simulation results are presented to validate the proposed estimation models. Based on these models, the detection period and the test size have been optimized to improve the performance of the overall fault-tolerance framework. Details are also provided about the proposed threshold-training and re-mapping methods.

III. Motivational Example

We simulate the training procedure of the VGG-11 network on the Cifar-10 dataset [32] for both fault-free training and on-line training with hard faults, and the results on the TestSet are shown in Fig. 2. The percentage of RRAM cells with stuck-at faults after fabrication is approximately 10% [23]. The endurance model is based on the published data that the endurance of cells obey a Gaussian distribution [4], and we set the mean endurance of cells to be equal to $5 \times 10^6$ [7], [8], [33]. Training in the absence of faults can classify the TestSet with 85.2% accuracy after 5 million iterations. However, when SA1 and SA0 faults are injected during training according to the above endurance model, the maximum accuracy can reach only 37%, and the accuracy further decreases if we continue with more training iterations. Moreover, if we need to train the RCS for another subsequent neural-computing application, which aggravates the impact of faults (i.e., the percentage of RRAM cells with faults may become 50%), we cannot achieve higher than 10% accuracy after training. Therefore, a fault-tolerant on-line training solution for RCS is required.

IV. Fault-Tolerant Training

The complete flow of the proposed fault-tolerant on-line training method is shown in Fig. 3. In prior work, a forward propagation phase is processed to obtain the actual output from the current neural network [5]. Then, the ideal output from the training dataset is compared with the actual output, and the difference is back-propagated to update network weights. In the proposed fault-tolerant training method, a threshold-training step (described in Section VII.A) is proposed after the back-propagation phase to enhance the lifetime by eliminating write operations on RRAM cells. Subsequently, after every fixed number of iterations, the proposed on-line fault detection method (described in Section V) is executed to update fault-free/faulty status of RRAM cells. A pruning step is carried out simultaneously to obtain the locations of zeros in the weight matrices based on network pruning [11]. When the percentage of newly detected faults is larger than a threshold, the
proposed re-mapping technique (described in Section VII.B) is applied. With the knowledge of the two distributions (the distribution of faulty cells and the distribution of zeros in the weight matrices), the re-mapping technique tolerates faults by utilizing both the detected fault distribution and the inherent sparsity of neural algorithms.

V. On-line Fault Detection

In this section, we first propose a fault detection method that uses quiescent-voltage comparison. Next, a modulo operation method is introduced to reduce hardware overhead. In order to achieve an acceptable level of performance with reduced test time, we propose an enhancement that applies testing only to selected cells.

The endurance of a cell is a random variable that obeys the Gaussian distribution [4]. For example, the simulation of the motivational example in Section III assumes that the distribution of endurance follows a Gaussian distribution with a mean of $5 \times 10^6$ and a variance of $1.5 \times 10^6$. The mean value is the average endurance of the RRAM cells. For each RRAM cell, the endurance indicates the maximum number of write operations that can be completed on this cell. Therefore, the likelihood of a fault depends on the number of write operations that have been completed on this cell. In the simulation, we accumulate the number of write operations that have been completed on each RRAM cell. Then, based on the endurance model, we calculate the probability that the cell will be faulty after the next write operation. Since there is no consensus yet on the spatial fault distribution in an RRAM crossbar, we employ the widely used uniform distribution [23].

A. Detection of Faults by Quiescent-Voltage Comparison

The flow of the proposed quiescent-voltage comparison method is described in Fig. 4. A read operation is first carried out to extract the RRAM crossbar values after training and store them off-chip. The SA0 and SA1 faults need separate detection procedures. The SA0 fault detection procedure consists of four steps. The first step is to write a fixed increment to all RRAM cells. We replace the traditional “Write 1” operation with a “Write $+\delta w$” operation, where $+\delta w$ is the fixed increment. In this way, we can recover the training weights of RRAM cells after testing. Second, test voltages are applied to one group of rows. Utilizing the crossbar structure, output voltages can be obtained at all column output ports concurrently. This parallel test method is therefore time-efficient. In the third step, control logic and a multiplexer are utilized to select an appropriate reference voltage. The reference voltages are set to be equal to the sum of previous values stored off-chip and the written increments. The last step involves comparisons between the actual outputs and the reference voltages. If a discrepancy exists, it denotes that at least one of the RRAM cell in the selected rows and columns cannot be updated correctly when we write an increment. This discrepancy indicates a SA0 fault. Because RRAM crossbars can be used in two directions, we can also apply test voltages to column input ports, and repeat the last three steps to derive the row information from the row output ports. After SA0 fault detection, a “Write $-\delta w$” operation is carried out for SA1 fault detection. The reference value is set equal to the increment value used for SA0 detection to enable the RRAM to recover its original training weights. The four-step test process for SA1 fault detection is similar to that for SA0 fault detection.

An example is presented in Fig. 5(a) to illustrate the test procedure. The crossbar here is a simplified schematic of a $10 \times 10$ RCS, and a square with a cross represents an actual defective cell. The test size is set to $5 \times 5$, which implies that the test voltages are applied to five rows in every test cycle. Therefore, to finish the row test, we need two test cycles. An additional two cycles for column test are required. Shaded squares are determined as being fault-free. In Fig. 5(a), the white squares denote the detected fault locations. As shown in this figure, 100% fault coverage is obtained, but some fault-free cells are determined to be faulty.

B. Reduction of Hardware Overhead with Modulo Operations

To tolerate write variance, RRAM cells in the test phase of RCS can be considered as multi-level resistance cells. The increment (reduction) written in testing phase is set to be larger than the variance. The number of resistance levels can be set to 8 [34].

Although the same voltage is applied to all the tested rows, the correct outputs may still be in a large range for different conductance ($g_{kj}$) combinations. To simplify the design of the reference voltage, we use conductance operations to map all possible voltages to a limited number of values. We choose 16 as the divisor for the modulo operations based on a trade-off between fault coverage and hardware overhead. In this way, only 16 reference voltages are needed, and the control logic is also simplified. Faults can still be detected unless 16 or more faults occur simultaneously in the tested area. As the divisor for the modulo operation increases, the fault coverage increases, but the hardware overhead increases. In order to apply modulo operations, we reuse the analog-to-digital converters (ADCs) on the output ports, transforming the analog output voltages to digital values. The $\text{mod} (2^n)$ operations can be realized by truncating the last $n$ bits of the dividend. A set of NAND gates can then be used for digital comparisons. The loss in information due to the modulo operations leads to a slight decrease in fault coverage.

C. Improvement of Performance by Selected-Cell Testing

To make the test method practical, we must ensure low test time, high fault coverage, and low false positives for fault
Fig. 5: Illustration of the quiescent-voltage comparison test method (test size = 5 \times 5). (a) Detect faults among all cells. (b) Detect faults among selected cells.

detection. Using the quiescent-voltage comparison method for only the selected cells helps us to achieve this goal. Cells that need testing are selected based on the knowledge that SA0 faults can occur only in cells with high resistance, and SA1 faults can occur only in cells with low resistance. The read operation at the beginning of the test phase provides the necessary information about resistance values. The test procedure involving selected cells is shown in Fig. 5(b). For example, considering the detection of SA0 faults, the pink squares denote the cells with high resistance. Since SA0 faults can occur only in these pink cells, there is no need to test the other cells. In this way, the test time may be reduced because no test is applied to the first column. The number of false detections is also decreased from 10 in Fig. 5(a) to 6 in Fig. 5(b).

VI. PERFORMANCE MODELS FOR FAULT DETECTION

Compared with traditional off-line fault detection methods, the proposed on-line fault detection method based on quiescent-voltage comparison reduces the high time overhead of fault detection. However, since this method tests multiple cells each time, some fault-free cells may be erroneously identified as being faulty. Moreover, with the modulo operations, some faulty cells may be incorrectly identified as being fault-free.

The performance of fault detection is influenced by two parameters: 1) the test size, and 2) the detection period. We define \( T_r \) (\( T_c \)) as the number of selected rows (columns) in each test cycle, and the test size is denoted by \( T_r \times T_c \). We define \( C_r \) (\( C_c \)) as the number of rows (columns) in the crossbar. Since the test phase is performed after a fixed number of iterations, we define the detection period \( t \) as the number of training iterations between two consecutive test phases.

When the test size is small and the detection period is short, we can locate the faults precisely in the detection phase, but the testing-time overhead becomes large. In order to analyze the trade-off between the testing time and the fault-detection performance, an analytical model is required. In this section, we first define the evaluation metrics. Next, we propose several models to analyze the influence of the two design parameters, and these models help us optimize the parameters and obtain a better trade-off among the evaluation metrics.

A. Evaluation Metrics

We use the statistical metrics of precision (PR) and recall (RE) to evaluate the percentage of incorrect test outcomes, i.e., a fault-free cell is determined to be faulty, or vice versa. Let \( TP \) refer to the number of faulty cells that are correctly identified as being faulty. These cells are referred to as CCF (Correctly Classified as Faulty) cells. Let \( FP \) be the number of fault-free cells that are erroneously identified as being faulty. A ICFF (Incorrectly-Classified as Fault-Free) cell is defined as a faulty cell that is incorrectly identified as being fault-free, and \( FN \) denotes the number of ICFF cells. Then \( PR = TP/(TP + FP) \), and \( RE = TP/(TP + FN) \).

In addition to precision and recall, test time is another important metric to evaluate the time overhead of the detection phase. We define the test time for a single test phase as the number of cycles used for fault detection in one test phase, and it is denoted as \( T_{single} \).

B. Models for Single Test Phase

Suppose that the percentage of RRAM cells containing faults is \( r \). When we use a small test size, the precision and recall are high, but the time overhead can be large. Without any loss of generality, we set \( T_r = T_c \) since the fault distributions are independent of the row or column directions [28]. To optimize the test size, we propose approximation models for precision and recall, and verify these models by simulations in Section VIII-B.

1) Recall Model: \( RE \) is calculated by \( TP/(TP + FN) \). The value of \( FN \) equals to the number of ICFF cells, and in the proposed method, the ICFF cells only occur when the actual output of a selected row (column) with faults is the same as the fault-free output. The proposed detection method contains two steps, namely the quiescent-voltage calculation and the modulo operation, and only the modulo operation contributes to the occurrence of false-negative errors and increase in \( FN \). In the quiescent-voltage calculation step, we test for SA0 and SA1 faults in two independent steps; hence in each step, all the target faulty cells generate deviations from the correct output voltages. In the modulo operation step, when multiple faults appear in the same row or column, the deviations may be counter-balanced, and this condition contributes to the increase in \( FN \).

During quiescent-voltage comparison, we first write a same value \( \delta w \) to every RRAM cell, then apply a same test voltage \( V_{test} \) to all the selected input ports, and calculate the computing results of the corresponding rows (columns). If an RRAM cell contains a stuck-at fault, the fault will cause a deviation of value \( V_{test} \times \delta w \) from the expected output voltage on the corresponding output port. Therefore, \( V_{test} \times \delta w \) is set as the least significant bit (LSB) of the ADC in the detection phase, and one SA fault in the selected row (column) causes a one LSB deviation on the digital output.

The accumulated digital output deviation \( d \) of a selected row (column) is equal to the number of faulty RRAM cells in the
selected row (column). For the modulo operation, we denote the modulus as \( m_d \). Therefore, a false negative error occurs if and only if \( d \) and 0 are congruent modulo \( m_d \), namely \( d \equiv 0 \pmod{m_d} \). If \( T_r \leq m_d \), no false-negative error will occur and the recall of the proposed method equals to 1. If \( T_r > m_d \), false negative errors occur only when a selected row (column) contains \( k \times m_d \) faulty cells (\( k=1,2,... \)).

From the perspective of each RRAM cell, when an RRAM cell in a selected row (column) contributes to \( F_N \), two conditions need to be satisfied. 1) This cell is faulty; 2) This cell is identified as a fault-free cell. Since the cell faults occur independently, these two conditions are independent. Therefore, the probability that a cell contributes to \( F_N \) can be calculated by the product of the probability of the above two parts. Suppose that the percentage of RRAM cells containing faults is \( r \), and the SA faults are independent and obey uniform distribution [28]. Let \( P(F_N_{cell}) \) denote the probability of an event that an RRAM cell in a selected row (column) contributes to \( F_N \), which can be calculated by:

\[
P(F_N_{cell}) = r \left( \frac{T_r - 1}{km_d - 1} \right)^{km_d - 1}(1 - r)^{T_r - km_d} \tag{2}
\]

From the perspective of a selected row (column), if the number of faulty cells equals \( km_d \), all the faulty cells are identified as the fault-free ones. From the perspective of a selected row (column), ICCF cells occur if and only if the number of faulty cells equals \( km_d \), and in these cases, all the faulty cells are identified as the fault-free ones. Therefore, the mathematical expectation of \( F_N \) in a selected row (column), denoted as \( E(F_N_{row}) \) (or \( E(F_N_{col}) \)), can be calculated by:

\[
E(F_N_{row}) = \sum_{k=1}^{\lfloor T_r/m_d \rfloor} km_d P(F_N_{row} = km_d) \tag{3}
\]

\[
= \sum_{k=1}^{\lfloor T_r/m_d \rfloor} km_d \left( \frac{T_r}{km_d} \right)^{km_d}(1 - r)^{T_r - km_d} \tag{4}
\]

\[
E(F_N_{col}) = \sum_{k=1}^{\lfloor T_r/m_d \rfloor} km_d P(F_N_{col} = km_d) \tag{5}
\]

\[
= \sum_{k=1}^{\lfloor T_r/m_d \rfloor} km_d \left( \frac{T_r}{km_d} \right)^{km_d}(1 - r)^{T_r - km_d} \tag{6}
\]

where \( P(F_N_{row} = km_d) \) and \( P(F_N_{col} = km_d) \) are the probability of the event that a row and a column contains \( km_d \) faulty cells, respectively.

Finally, for the entire crossbar, if we ignore the interaction between column test and row test, the mathematical expectation of \( F_N \) can be approximately calculated by:

\[
E(F_N) = E(F_N_{row})C_c(C_r/T_r) + E(F_N_{col})C_r(C_c/T_c) \tag{7}
\]

\( TP \) denotes the number of faulty cells that are correctly detected, which equals to the number of faulty cells minus \( F_N \). Therefore, we use an approximation model to calculate the expectations of \( TP \) and \( RE \) by:

\[
E(TP) = rC_cC_r - E(F_N) \tag{8}
\]

\[
E(RE) = E(TP)/(TP + F_N) \approx \frac{rC_cC_r - E(F_N)}{rC_cC_r} \tag{9}
\]

where \( E(F_N) \) is calculated using the model in (7).

2) Precision Model: As shown in the expression \( PR = TP/(TP + FP) \), precision (PR) is mainly influenced by \( FP \), where \( FP \) increases when a fault-free RRAM cell is detected as a faulty cell. The proposed quiescent-voltage comparison method tests multiple selected RRAM cells simultaneously to reduce the test time overhead. Compared with the cell-by-cell test method used in off-line detection, the proposed method cannot always distinguish the faulty cells from the fault-free cells if multiple faults occur in a tested area, as shown in Fig. 5, and this reduces precision.

When an RRAM cell contributes to the increase in \( FP \), we refer this cell as an ICF (Incorrected Classified as Faulty) cell. In the proposed quiescent-voltage comparison method, a ICF cell occurs if and only if: 1) the cell is fault-free; 2) at least one faulty cell is located in the same selected row; and 3) at least one faulty cell locates in the same selected column. Since the faults are independent in the RRAM array, the above three conditions (events) are also independent. Therefore, the probability that a cell contributes to an increase in \( FP \), which is denoted as \( P(FP_{cell}) \), can be calculated by the product of the probability of three events:

\[
P(FP_{cell}) = (1 - r) \times [1 - (1 - r)^{T_r - 1}] \times [1 - (1 - r)^{T_c - 1}] \tag{10}
\]

where \( (1 - r) \) denotes the probability that this cell is fault-free, \([1 - (1 - r)^{T_r - 1}] \) denotes the probability that the other \( T_r - 1 \) cells in the same selected row contain at least one faulty cell, and \([1 - (1 - r)^{T_c - 1}] \) denotes the probability that the other \( T_c - 1 \) cells in the same selected column contain at least one faulty cell.

To further analyze the mathematical expectation of \( FP \) for a given test size, we need to calculate the probability for all situations, i.e., \( P(FP = x) \) where \( x = 1, 2, \text{etc.} \). However, enumerating all the situations for large \( FP \) is difficult, and therefore we use an approximation to simplify the model.

We add an assumption that the ICF cells are independent, and derive an approximation model based on this assumption to estimate the mathematical expectation of \( FP \) in a tested area. This assumption is stronger than the practical situation because the distribution of the ICF cells is correlated with the distribution of faulty cells. However, since faults are independent, the correlations introduce little deviation when we calculate the mathematical expectation \( E(FP) \). The influence of this assumption is verified by simulation results in Section VIII-B.

Based on the independent assumption of ICF cells, we can derive the approximation model for \( FP \) in an RRAM array using (10):

\[
E(FP) = C_r \times C_c \times P(FP_{cell}) \tag{11}
\]

\[
= C_r C_c (1 - r) [1 - (1 - r)^{T_r - 1}] [1 - (1 - r)^{T_c - 1}] \tag{12}
\]
The precision of detection, \( PR \), can be approximately calculated using \( TP \), \( FP \), and \( FN \):

\[
E(\text{PR}) = E\left(\frac{TP}{TP + FP}\right)
\]

\[
\cong \frac{rC_rC_c - E(\text{FN})}{rC_rC_c - E(\text{FN}) + E(\text{FP})}
\]

where \( E(\text{FN}) \) and \( E(\text{FP}) \) are calculated by (7) and (12), respectively.

3) Test Time: The test time is computed as: \( \tau_{\text{single}} = \lceil C_r/T_r \rceil + \lceil C_c/T_c \rceil \). We define \( E_c \) (\( E_r \)) as the number of rows (columns) that contain RRAM cells with high or low resistance; these cells are likely to be faulty cells. Since we need to perform testing only in these selected rows and columns using the proposed selected-cell testing, the test time is reduced, which can be calculated by:

\[
\tau_{\text{single}} = \lceil E_r/T_r \rceil + \lceil E_c/T_c \rceil
\]

C. Extension of Models to the Entire Training Procedure

The discussion in Section VI-B is based on an assumption that the percentage of the newly generated faults in a test phase is \( r \). We perform fault detection after a fixed number of training iterations, namely the detection period \( t \), as shown in Fig. 3. When we use larger \( t \), a larger number of write operations are performed on each RRAM cell before each test, which causes more SA faults due to the limited endurance. Therefore, given the endurance model of RRAM cells, \( r \) is a variable related to the detection period \( t \). Assume that \( P(\cdot) \) is the probability distribution function of the endurance of an RRAM cell, and the probability distribution derives from the endurance model for the off-line testing, and \( F(\cdot) \) is the cumulative distribution function. For example, \( F(10^6) = P(x \leq 10^6) \) means the probability that the endurance of a cell is less than \( 10^6 \). Therefore, if the number of writing operations that have been performed on an RRAM cell before the \( i \)th test phase is \( W(i) \), we can calculate the probability of a cell that becomes faulty between the \( i \)th and \((i + 1)\)th test phase as:

\[
P_{\text{NewFault}} = P(W(i) < x \leq W(i + 1)) = F(W(i + 1)) - F(W(i))
\]

Since the SA faults caused by limited endurance are independent on different RRAM cells, we have \( r = P_{\text{NewFault}} \).

In the original training method, more than 99% of RRAM cells need to be modified in a training iteration. Therefore, for almost every RRAM cell, a write operation is performed in every iteration, so we can approximately calculate \( W(i + 1) = W(i) + t \). In this way, the percentage of newly generated faults in a test phase \( r \) can be calculated by:

\[
r = F(W(i + t) - F(W(i))
\]

Based on the models proposed in Section VI-B, when we use the same test size, higher \( r \) leads to lower precision, which means that the test period \( t \) cannot be too large. However, lower \( t \) means we need to carry out test phase more frequently, which leads to higher time overhead during the network training. To obtain a better trade-off, we regard \( r \) as a variable determined by (17), and introduce the influence of \( t \) into the precision and recall models proposed in Section VI-B. Specifically, given the endurance model and the average number of write operations that have been performed on the RRAM cells, \( r \) is determined by the design parameter \( t \) using (17). After that, by substituting \( r \) into the models in Section VI-B, the precision and recall with specific test period \( t \) and test size \( s \) can be estimated.

For the time overhead, \( \tau_{\text{single}} \) evaluates the time overhead of a single test phase. However, since the test phase is performed after every fixed number of iterations, the time overhead during the entire training procedure is influenced by not only \( \tau_{\text{single}} \), but also the number of iterations between two consecutive test phases. Therefore, we accumulate all the test time together to evaluate the overall time overhead, which is denoted as \( \tau_{\text{accumulated}} \). If the number of iterations for an entire NN training is \( N_{\text{Iteration}} \), the overall time consumption for the test phases \( \tau_{\text{accumulated}} \) can be calculated based on (15):

\[
\tau_{\text{accumulated}} = \tau_{\text{single}} \times \frac{N_{\text{Iteration}}}{t}
\]

\[
= \left(\lceil E_r/T_r \rceil + \lceil E_c/T_c \rceil \right) \times \frac{N_{\text{Iteration}}}{t}
\]

In this way, the three performance factors \{\( PR, RE, \tau_{\text{accumulated}} \)\} with specific \( \{t, s\} \) can be estimated, which help us optimize the choice of \( \{t, s\} \).

VII. On-line Training

In this section, we first reduce the occurrence of new faults using the threshold training method, as described in Section VII.A. Then, a re-mapping method is proposed in Section VII.B to utilize existing faults as unchanged functional weights.

A. Threshold Training

We first introduce the threshold-training method to reduce the number of write operations in each training iteration. \( w_{i,j}^{n}(t) \) is the weight in the \( i \)th row and \( j \)th column of layer \( n \), and \( t \) is the iteration count. In the back-propagation step of on-line training, the weight in the next iteration \( w_{i,j}^{n}(t + 1) \) is determined by:

\[
w_{i,j}^{n}(t + 1) = w_{i,j}^{n}(t) + LR \ast \delta w_{i,j}^{n}(t)
\]

\[
\delta w_{i,j}^{n}(t) = x_{i}^{n}(t) \ast f'(y_{j}^{n}(t)) \ast \delta y_{j}^{n}(t)
\]

where \( x_{i}^{n}(t) \) is the input of \( w_{i,j}^{n}(t) \) from the previous layer in the forward-propagation step, \( f'(y_{j}^{n}(t)) \) is the derivative of the output in the forward-propagation step. \( LR \) is the learning-rate parameter used to control the training speed, which is first set to a large value and gradually decreased during training.

To reduce the occurrence of faults, we analyze the distribution of \( \delta w \) among all the weights in one iteration. For approximately 90% of the weights, \( \delta w \) is less than \( 0.01 \delta w_{\text{max}} \), where \( \delta w_{\text{max}} \) denotes the maximum \( \delta w \) in this iteration. When we consider the endurance problem, a small value of \( \delta w \) contributes only slightly to network training but reduces the lifetime of the RRAM cell.

Based on this observation, we only write to an RRAM cell with a large \( \delta w \). The proposed threshold-training method
is shown in Algorithm 1. Line 1 is the forward-propagation step processed on RRAM arrays, while lines 2-3 indicate the back-propagation step used to obtain $\delta w$. For lines 4-13, if $\delta w$ is less than the training threshold, it will be reduced to zero. Therefore, the corresponding RRAM cell can avoid the write operation. The threshold is set to 0.01 of the maximum value of $\delta w$, which improves the average lifetime of each RRAM cell by approximately $15 \times$. Some on-line training RCSs perform the calculation of $\delta w$ in software [5]; hence, the proposed threshold-training method can be integrated into the training algorithm shown in Algorithm 1. If the calculation is implemented in hardware [6], a threshold comparator is required to check whether a change is large enough and thereby avoid writing the small changes.

Fig. 6 shows the training curve comparison of VGG-11 on Cifar-10. The experimental set-up is described in Section VIII.A. The loss function is a surrogate of NN accuracy, which is widely used to reflect the training progress of NN [35], [36]. As shown in Fig. 6, the training curves are similar, which shows using threshold training has negligible impact on the training curve.

From a software perspective, threshold training increases the training time. Since the number of RRAM write cycles depends on the number of write peripheral circuits and the write scheme, we use the number of training iterations to compare the time overhead caused by threshold training. We test the method on (1) a $784 \times 100 \times 10$ NN on the MNIST dataset [37] and (2) the VGG-11 network on the Cifar-10 dataset. Compared to the original training method, the number of training iterations is increased 1.2 times. Therefore, the eventual number of weight update cycles is 1.2 times more compared to the original training method.

![Algorithm 1: Threshold-training Algorithm](image)

### B. Fault-Tolerant Re-mapping Method

Although threshold training can reduce the impact of write operations, new hard faults induced by limited endurance will inevitably occur during the training procedure, and impact the network performance (i.e., the recognition accuracy).

At the software level, researchers have noted that the weights in a neural network contain a large amount of redundant information. Therefore, pruning methods have been proposed to fix more than 50% of the weight values to zero during training [11]. This finding motivates us to use the inherent sparsity of a neural network to tolerant the SA0 faults in an RRAM cell. This target can be achieved by re-ordering the columns/rows of the weight matrix, and map the zeros in the weight matrices to RRAM cells with SA0 faults.

The challenge in designing such an exchange method lies in the inherent connection between matrices. As shown in Fig. 7, from a software perspective, a neural network consists of multiple cascaded layers. Therefore, the outputs of the RRAM crossbar are connected to the inputs of another RRAM crossbar through peripheral neuron modules [2]. Consequently, if we independently exchange the rows or columns in each weight matrix with $M$ neurons, an $M$-to-$M$ routing module is required to connect different RRAM crossbars, which introduces high area and energy overhead. To address this problem, we only consider the re-ordering of neurons in this work. In other words, when the $i$th and $j$th columns of the $(n-1)$th layer’s weight matrix are exchanged, the $i$th and $j$th rows of the $n$th layer will also be exchanged in a corresponding manner. From a software perspective, if we regard the neural network as a weighted graph, the re-ordering of neurons with connected weights will lead to an isomorphic network whose interconnection structure is the same as that of the original network.

After the fault-detection step, there are two networks: 1) an $N$-layer pruned network $P$ provided by the pre-trained result from software training; 2) an $N$-layer fault distribution network $F$, obtained after each fault detection phase described in Section V. Specificaly, $P = \{P^{(1)}, P^{(2)}, ..., P^{(N)}\}$, where $P^{(n)} = \{p^{(n)}_{i,j}\}$ is a 2-dimensional weight-pruning matrix for the $n$th layer. If the weight in a neural network can be pruned, the corresponding value of $p^{(n)}_{i,j}$ is set to 0; otherwise, the value of $p^{(n)}_{i,j}$ is set as $\infty$. $F = \{F^{(1)}, F^{(2)}, ..., F^{(N)}\}$, where $F^{(n)} = \{f^{(n)}_{i,j}\}$ is a 2-dimensional fault-distribution matrix for the $n$th layer. If an SA0 fault occurs on RRAM cell, the
corresponding value of \( f_{i,j}^{(n)} \) is set to 0; if an SA0 fault occurs, the corresponding value of \( f_{i,j}^{(n)} \) is set to 1; otherwise, the value of \( f_{i,j}^{(n)} \) is set as \( \infty \).

To describe whether an SA0 fault in \( F \) is tolerated (reused) by the inherent zeros in the weight matrices, an Error Set \( E = \{ e \} \) is defined as the set of address groups, where \( e = \{ i, j, n \} \) is the address of a weight that satisfies: \( e \in E \) iff \( (f_{i,j}^{(n)} \neq 0 \& f_{i,j}^{(n)} \neq \infty) \). The optimization target is to minimize the number of elements of \( E \). Therefore, the distance between \( P \) and \( F \) can be defined as the number of elements of \( E \), i.e.,

\[
\text{Dist}(P, F) = |E| \tag{22}
\]

We can similarly define \( E^{(n)} \) as the error set corresponding to layer \( n \). The distance \( \text{Dist}(P, F) \) between \( P \) and \( F \) is the sum of distances between \( P^{(n)} \) and \( F^{(n)} \), i.e.,

\[
\text{Dist}(P, F) = \sum_{n=1}^{N} \text{dist}(P^{(n)}, F^{(n)}) = \sum_{n=1}^{N} |E^{(n)}| \tag{23}
\]

Based on these definitions, the optimization target of our neuron re-ordering algorithm can be described as follows. Given \( F \) and \( P \) with \( \|F\| = \|P\| \), \( O \) is the set of the networks that \( P \) can be re-ordered into; the optimization result is to find \( P_{opt} \in O \) that has the minimum distance from \( F \), i.e.,

\[
P_{opt} = \arg\min_{X \in O} \{ \text{Dist}(X, F) \} \tag{23}
\]

The neuron re-ordering problem can be mapped to a set of Knapsack problems, hence it is NP-hard. We use a genetic algorithm to iteratively optimize the order of neurons layer by layer. For each layer, we randomly exchange two neurons and evaluate the change in the cost function \( \text{Dist}(P, F) \).

---

**VIII. SIMULATION RESULTS**

**A. Experimental Setup**

1) Fault Model: Since there is no consensus yet on the spatial fault distribution in an RRAM crossbar, we evaluate our approach using the widely-used uniform distribution [23]. The fabrication defects cause 10% of the RRAM cells to have stuck-at faults [23]. Two endurance models for RRAM cell are used to target both low-endurance RRAM cell and high-endurance RRAM cell. The mean endurance of low-endurance RRAM cell is set at \( 5 \times 10^6 \) write operations and the distribution of endurance obeys a Gaussian distribution [4] with a variance of \( 1.5 \times 10^6 \). For the high-endurance model, we assume that the distribution of endurance follows a Gaussian distribution with a mean of \( 10^6 \) and a variance of \( 3 \times 10^7 \). For each RRAM cell, the likelihood of a fault depends on the number of write operations. We verify the proposed approximation models for detection precision and recall by comparing the simulation results with

2) Benchmarks: We evaluate the proposed method on a modified VGG-11 deep neural network for the Cifar-10 dataset [32]. The VGG-11 network is modified to match the input size of Cifar-10 dataset, which contains 8 Conv layers and 3 FC layers. The total weight amount is 7.66M and the complexity is 137M operations. We have used commonly used regularization techniques to confirm that the presented results do not suffer from the overfitting problem. The network converges at around \( 5 \times 10^6 \) iterations, and the recognition accuracy is \( 85.2\% \) on \( TestSet \), which is set as the ideal case for training without faults. In the simulations of the on-line detection part, without loss of generality, the number of rows and columns in the crossbar are assumed to be equal.

**B. Model Verification**

We verify the proposed approximation models for detection precision and recall by comparing the simulation results with...
and the estimated values. The crossbar size is set to 256. Assume that 10% of the cells are faulty and 30% of the cells are in a high-resistance state [28]. With a large test size, the approximation in Section VI introduces more deviation into the proposed model. To evaluate the difference between the estimated value from the proposed model and the actual simulated value, we first set the test size to a large value, i.e., \( T_r = T_c = 100 \). We use the uniform fault distribution, and randomly insert 100 faults in the simulation. As shown in Fig. 8, the deviation between the average simulated values and the estimated values is less than 0.001, which means that the relative error is less than 1%.

To evaluate the accuracy of the approximation model when using different test sizes, we iterate the simulation where \( T_r = T_c \) are set from 1 to 100, and each simulation is run with 100 random samples of fault distributions. Even with \( T_r = T_c = 100 \), the worst-case recall is larger than 99.5%. Therefore, we only evaluate the precision model with different test sizes. As shown in Fig. 9, we verify the models on two testing methods: a basic test method without using the prior knowledge about the cells’ resistances, and the selected-cell testing method proposed in Section VC. The proposed models match the simulation results in both methods with all test sizes. The average deviation between the estimated values and the simulation values is less than 1%.

C. Effectiveness and Trade-off Analysis of Fault Detection in One Phase

There are interesting trade-offs between test time, precision, and recall for the on-line detection method. In the detection process, we can use a smaller test size to increase the precision and recall, but the test time will also increase. Fig. 10 shows the trade-off results for detecting faults in RRAM crossbars. The percentage of faulty cells is set to 10%, and the RRAM crossbar sizes used in the simulations range from 128 \( \times \) 128 to 1024 \( \times \) 1024. This range is motivated by the current state of fabrication technology and the prospects for future advances. The fault detection recall values increase slowly as the test time increases. Since the recall is always larger than 99%, it is acceptable for the subsequent neural computation steps. On the other hand, for a given precision, the test time grows linearly with crossbar size. Therefore, large RRAM crossbars need longer test time to achieve satisfactory performance.

In order to demonstrate the advantage of testing only among selected cells, we compare the performance of testing among all the cells with the performance gained by testing a subset of cells in terms of test time. The performance results show that, with this improvement, the precision increases significantly from around 50% to 77%, while the recalls of both methods remain above 99%.

In order to demonstrate the effectiveness of the proposed on-line testing method, we compare the test time of the proposed method with the sneak path technique [29]. Even though the sneak path technique incorporates test parallelism by testing 13 adjacent RRAM cells simultaneously, the time complexity still remains proportional to the array size. For an RRAM crossbar of size 128 \( \times \) 128, a 14\( \times \) speed-up can be obtained using the proposed method. For larger RRAM crossbar arrays, more than 14\( \times \) speed-up can be obtained.

D. Effectiveness and Trade-off Analysis of Fault Detection in Multiple Phases

We further evaluate the performance of the on-line detection method in training an entire NN. Besides the test size, another parameter, i.e. the detection period, is also considered and optimized. The test size influences the precision, recall, and test time in one test phase, as discussed in Section VIII-C. The detection period is the number of training iterations between two successive detection phases. When the detection period is small, the number of newly generated faulty cells is small in the next detection phase. Therefore, the precision and recall are high with only a small amount of newly generated faulty cells. However, a small detection period leads to a long time, so there are trade-offs between the overall test time, precision, and recall. Since the recall is always larger than 90%, which is acceptable for the subsequent neural computation steps, we focus on optimizing the precision in the subsequent experiments. With a quiescent-voltage comparison using modulo operations, the precision and test time results are shown in Fig. 11. The precision is calculated by the proposed approximation model (14), and the test time is calculated using (19). When we use smaller test size and smaller detection period, the precision becomes larger but the test time is also enlarged. Therefore, there is a trade-off between test time and precision, as shown in Fig. 12. When the test size is 5\( \times \)5 and the detection period is 5000, 85% precision can be obtained with 4.1\( \times \)10\(^5\) test cycles.

In order to evaluate the effectiveness of the proposed flow, we present the time needed for training a complete VGG-11 network on an RRAM-based computing system for 5,000,000 training iterations with the proposed fault-tolerant training techniques in Table I. For the write latency, related architectural designs [2], [22] set the clock frequency to 500 MHz; however, chip demonstrations typically use 10 MHz to 100 MHz clock frequencies [19], [38], [39]. Therefore, in Table I, we set the clock frequency to 500 MHz, 100 MHz,
and 10 MHz as three examples to estimate the time overhead. We employ the widely used column-by-column write scheme [5], [22] to reduce the time needed for write operations, which implies that the RRAM cells in the same column are written to simultaneously. For the read circuit design, ADCs that can read the computation results in one cycle [3], [13], [40], [41] are used in our simulation. The off-chip data access speed is from [42]. Fig. 13 shows the breakdown when the clock frequency is 100 MHz. The detection phase consumes 3% of the training time. The time overhead is small due to two reasons. First, the detection phase is performed every 5000 training iterations. Second, due to the convolutional structure, a RRAM crossbar in a Conv layer is reused 64 to 1024 times in the forward propagation and back propagation phases, but a crossbar needs to be tested only once in a detection phase.

### E. Results of Entire Fault-tolerant Training

We first evaluate the proposed method on a fully-connected NN using the MNIST dataset. The network size is 784 × 100 × 10, and the low-endurance model is used. Fig. 14 shows the accuracy for TestSet during the training phase. Influenced by the hard faults, the maximum accuracy of the original training method is 91.53% (the orange line). The proposed threshold training method can increase the accuracy to 97.47% (the grey line), which is similar to the ideal case accuracy of 97.89% (the blue line). However, the 3-layer fully-connected network is too simple compared with state-of-the-art CNNs such as
VGG. Therefore, we further evaluate the proposed methods on RRAM-based CNN applications.

For RRAM-based CNN, some researchers implement both Conv layers and FC layers on RCS [43], while other researchers only implement FC layers on RCS [13]. Software level results have shown that the fault tolerance capability and sparsity of the Conv layer are much lower than that of the FC layer [11]. Therefore, we evaluate the proposed methods in two cases. (1) An entire-CNN case: all the layers of the VGG-11 network are mapped onto an RCS. (2) A FC-only case: only the FC layers are mapped onto an RCS.

We evaluate the original training method using multiple endurance models and initial fault ratios in both cases. Our simulation results show that Conv layers are sensitive to hard faults, and thus, the accuracy of CNN is approximately 10%, if more than 20% of the RRAM cells have hard faults. However, for the FC-only case, the accuracy decreases only when the percentage of faulty cells is larger than 50%.

Fig. 17(a) presents the accuracy on TestSet for the entire-CNN case. The mean endurance value of RRAM cells is set to $5 \times 10^6$. Without fault-tolerant training, the accuracy can drop to only 10%, and the largest accuracy obtained during training is also lower than 40% (the orange line). The threshold-training method (the grey line) can increase the peak training accuracy to 83% on the Cifar-10 dataset, which is comparable to the fault-free software-based training results (the blue line). However, since the fault tolerance and sparsity of Conv layer is low, the proposed fault-detection and re-mapping steps cannot further improve the accuracy (the yellow line).

Fig. 15 shows the percentage of faulty cells during training of the first network for the entire-CNN case. Since more than 90% of write operations can be eliminated by the threshold training method, the percentage of faulty cells after training can be reduced from 37% to less than 1%. Fig. 16 shows the comparison on the basis of the number of write operations. The traditional training method writes into nearly every cell in every iteration during training. When threshold training method is used, the number of write operations can be reduced to 6% of that of the traditional training method on average.

We further evaluate the effect of threshold-training method for different endurance models. When a high-endurance model with mean value $10^8$ is used, the original method can train the RCS for approximately 10 times. Since the average number of write operations in the threshold-training method can be reduced to only 6% of that of the baseline method, the threshold-training method can train the RCS for more than 150 times. For the endurance model with a lower mean value like $10^7$, the percentage of faulty cells after the first training phase is approximately 14% using the original method; while in the second training phase, the training phase does not converge. The threshold-training method can successfully train the network with 14% of the cells being faulty; therefore, the RCS can be trained for approximately 27 times.

The FC-layers exhibit large inherent fault tolerance. However, the number of faulty cells increases after several training phases, and this will eventually render the chip nonfunctional. Therefore, for the FC-only case, we focus on the scenario in which the RCS has been trained multiple times and contains a large number of initial faults before the subsequent training commences. We set the mean endurance value to $10^8$, but reduce the remaining endurance of RRAM cells to mimic the case in which the RCS has already been trained multiple times. Fig. 17(b) shows the accuracy on TestSet for the extreme case in which the percentage of RRAM cells with hard faults before training is approximately 50%. With such a large number of initial hard faults, the peak accuracy of traditional on-line training method (the orange line) is only 63%. The threshold-
training method (the grey line) has negligible impact in this case because it can only reduce the occurrence rate of new faults but cannot tolerate the existing faults. The proposed fault-detection and re-mapping steps (the yellow line) can identify the hard faults in RRAM cells and tolerate them in the RCS by utilizing the sparsity of neural-computing algorithms. Based on the proposed fault-tolerant on-line training flow, the accuracy can be increased back to 76%. The accuracy loss decreases when the percentage of the initial faulty cells is low. For example, when the percentage of the initial faulty cells is 20%, the accuracy loss of the proposed fault-tolerant method is less than 1%.

IX. CONCLUSIONS

We have presented a fault-tolerant training method for an RRAM-based neural computing system to reduce the impact of hard faults in RRAM cells. An on-line fault-detection phase has been proposed to obtain the distribution of faulty cells during training by using a quiescent-voltage comparison method. Approximation models are proposed to analyze and optimize the performance of the proposed test methods. In the training phase, threshold training and heuristic re-mapping based on neuron re-ordering are proposed to tolerate faults using the inherent sparsity of neural networks. Simulation results show that the accuracy in the presence of faults can be restored to the original training accuracy for a fault-free scenario.

In future work, we will combine the proposed algorithm-level methods with related hardware-level methods [20], [28] to further improve the fault tolerance.

REFERENCES


Fig. 17: Accuracy of fault-tolerant on-line training method with different endurance models on the Cifar-10 dataset.
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2855145, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems


**Lixue Xia (S’14)** received the B.S. degree in Electronic Engineering from Tsinghua University, Beijing, in 2013. He is currently pursuing his Ph.D. degree in Department of Electronic Engineering, Tsinghua University. His research mainly focuses on energy efficient hardware computing system design, and neuromorphic computing system based on emerging Non-volatile devices. He has received the Best Paper Nomination on DAC 2017.

**Mengyun Liu** received the B.S. degree in Electrical and Computer Engineering from Shanghai Jiao Tong University, Shanghai, in 2016. She is currently pursuing her Ph.D. degree in Electrical and Computer Engineering in Duke University. Her research interests include testing for resistive RAM (RRAM) and improving fabrication testing with machine learning techniques.

**Xuefei Ning** received the B.S. degree in Electronic Engineering from Tsinghua University, Beijing, in 2016. She is currently pursuing her Ph.D. degree in Department of Electronic Engineering, Tsinghua University. Her research mainly focuses on deep learning and robustness of neural network.

**Krishnendu Chakrabarty (F’08)** received the B. Tech. degree from the Indian Institute of Technology, Kharagpur, in 1990, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively. He is now the William H. Younger Distinguished Professor and Department Chair of Electrical and Computer Engineering, and Professor of Computer Science, at Duke University. Prof. Chakrabarty is a recipient of the National Science Foundation CAREER award, the Office of Naval Research Young Investigator award, the Humboldt Research Award from the Alexander von Humboldt Foundation, Germany, the IEEE Transactions on CAD Donald O. Pederson Best Paper Award (2015), the ACM Transactions on Design Automation of Electronic Systems Best Paper Award (2017), and over a dozen best paper awards at major conferences. He is also a recipient of the IEEE Computer Society Technical Achievement Award (2015), the IEEE Circuits and Systems Society Charles A. Desoer Technical Achievement Award (2017), the Semiconductor Research Corporation Technical Excellence Award (2018), and the Distinguished Alumnus Award from the Indian Institute of Technology, Kharagpur (2014). He is a Research Ambassador of the University of Bremen (Germany) and a Hans Fischer Senior Fellow at the Institute for Advanced Study, Technical University of Munich, Germany. He is a 2018 recipient of the Japan Society for the Promotion of Science (JSPS) Fellowship in the “Short Term S: Nobel Prize Level” category.

Prof. Chakrabarty’s current research projects include: testing and design-for-testability of integrated circuits and systems; digital microfluidics, biochips, and cyberphysical systems; data analytics for fault diagnosis, failure prediction, anomaly detection, and hardware security; neuromorphic computing systems. He is a Fellow of ACM, a Fellow of IEEE, and a Golden Core Member of the IEEE Computer Society. Prof. Chakrabarty served as the Editor-in-Chief of IEEE Design Test of Computers during 2010-2012 and ACM Journal on Emerging Technologies in Computing Systems during 2010-2015. Currently he serves as the Editor-in-Chief of IEEE Transactions on VLSI Systems. He is also an Associate Editor of IEEE Transactions on Biomedical Circuits and Systems, IEEE Transactions on Multiscale Computing Systems, and ACM Transactions on Design Automation of Electronic Systems.

**Yu Wang (S’05-M’07-SM’14)** received his B.S. degree in 2002 and Ph.D. degree (with honor) in 2007 from Tsinghua University, Beijing. He is currently a tenured Associate Professor with the Department of Electronic Engineering, Tsinghua University. His research interests include brain inspired computing, application specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology. Dr. Wang has authored and coauthored over 200 papers in refereed journals and conferences. He has received Best Paper Award in FPGA 2017, NVMSA 2017, ISVLSI 2012, and Best Poster Award in HEART 2012 with 8 Best Paper Nominations (DAC 2017, ASPDAC 2016, ASPDAC 2014, ASPDAC 2012, 2 in ASPDAC 2010, ISLPED 2009, CODES 2009). He is a recipient of IBM X10 Faculty Award in 2010. Yu Wang also received The Natural Science Fund for Outstanding Youth Fund in 2016. Dr. Wang served as TPC chair for ISVLSI 2018, ICFPT 2011 and Finance Chair of ISLPED 2012-2016, Track Chair for DATE 2017-2018 and GLSVLSI 2018, and served as program committee member for leading conferences in these areas, including top EDA conferences such as DAC, DATE, ICCAD, ASP-DAC, and top FPGA conferences such as FPGA and FPT. Currently he serves as Co-Editor-in-Chief for ACM SIGDA E-News, Associate Editor for IEEE Transactions on CAD, IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), and Journal of Circuits, Systems, and Computers, and Special Issue Editor for Microelectronics Journal. He also serves as guest editor for Integration, the VLSI Journal and IEEE Transactions on Multi-Scale Computing Systems. He has given 70 invited talks and 2 tutorials in industry/academia. He is now with ACM Distinguished Speaker Program. He is an ACM/IEEE Senior Member.