Journal of Electronic Testing (2022) 38:145–163
https://doi.org/10.1007/s10836-022-06001-2

# FAMCroNA: Fault Analysis in Memristive Crossbars for Neuromorphic Applications

**Dev Narayan Yadav**[1] · **Phrangboklang Lyngton Thangkhiew**[2] · **Kamalika Datta**[3] · **Sandip Chakraborty**[1] · **Rolf Drechsler**[3,4] · **Indranil Sengupta**[1,5]

Received: 2 October 2021 / Accepted: 11 April 2022 / Published online: 13 May 2022

**Abstract**
Resistive memories have drawn the attention of researchers due to their low power and single-cycle computation of vector-matrix multiplication (VMM), which is the main operation performed in neural networks. For performing VMM, one of the most desirable architectures is the memristor crossbar that has several advantages over other memory technologies, viz. in-memory computation, low power, and high density. However, faults present in the crossbar can introduce errors in the inference process during neuromorphic computations. Existing methods to handle faults using retraining and remapping incur overheads in terms of hardware, power, and delay. In this paper we explore and analyze the impact of faults on memristor-based crossbar for overall inference accuracy. We have observed that the accuracy is not significantly affected in the presence of a limited number of faults. Also, the inference quality and effect of faults depend on the number of neural network layers and storage resolution of memristors present in the crossbar. The introduced approach works in three phases, fault tolerance analysis, high-level fault detection, and low-level fault detection. In the first phase, we analyze the fault tolerance capability of the crossbar, which identifies how many faults can be tolerated for a given application. In the second phase, we estimate the percentage of faults, and if it is below a threshold the third phase can be skipped. In the third phase, an efficient method to determine the exact location of the faults is used. The proposed method is capable of performing parallel operations, thus requiring fewer read/write steps as compared to existing works. The proposed approach requires $O(N)$ read/write operations as compared to $O(N^2)$ operations required in existing works.

**Keywords** Fault diagnosis · Fault tolerance · Memristor crossbar · Neural network · Stuck-at-faults

## 1 Introduction

Neural networks exhibit properties that imitate the brain and demonstrate significant performance improvements in various applications such as image, audio, and video

Responsible editor: L. M. Bolzani Poehls

✉ Dev Narayan Yadav
dev.narayan@iitkgp.ac.in

Phrangboklang Lyngton Thangkhiew
phrangboklang.thangkhiew@iiitg.ac.in

Kamalika Datta
kamalika.datta@dfki.de

Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

Rolf Drechsler
drechsler@uni-bremen.de

Indranil Sengupta
indranil.sengupta@jisuniversity.ac.in

[1] Computer Science & Engineering, Indian Institute of Technology Kharagpur, Kharagpur, India

[2] Computer Science & Engineering, Indian Institute of Information Technology Guwahati, Guwahati, India

[3] German Research Centre for Artificial Intelligence, DFKI, Bremen, Germany

[4] Institute of Computer Science, University of Bremen, Bremen, Germany

[5] Computer Science & Engineering, JIS University, Kolkata, India

recognition. The major computation performed in any neural network, both during the training and classification phases, is the *Vector-Matrix Multiplication* (VMM). The computation of VMM incur high overheads in conventional architectures both in terms of power consumption and latency.

Resistive memory technologies such as Memristors or Resistive Random Access Memory (ReRAM) possesses both storage and logic operation capability in a single cell. We can use a memristor crossbar to directly perform VMM operation in a single cycle and hence accelerate neural network operations. These crossbars do not require explicit data transfer between CPU and memory [34, 41]. But we require auxiliary circuitry like analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) that incur both space and energy overheads [3, 15, 27, 29].

In neuromorphic applications, training is an important and time-consuming phase. Prior to using the memristor crossbar as a classifier, it must be trained whereby the matrix coefficients (i.e., the memristor conductance values) get determined. This is generally performed using an off-line training phase using a given training set. For large crossbars this is a tedious and time-consuming process and may take days or even months. To accelerate the process, we can use *chip-in-the-loop* learning method [32], where the neuromorphic hardware is used to speed up the calculation. However, the focus of the current work is to analyze the impact of faults on neural network performance and hence we have used an off-line training method.

There are several factors like variation in conductance values, operational faults, fabrication defects, etc. [5, 17, 18], that degrade the performance of a memristor-based neuromorphic system. Various approaches have been proposed in the literature [6, 16, 23, 24, 35, 36, 38, 42, 43] that try to minimize the performance degradation in the presence of faults.

Some prior works in the literature include methods like retraining, row flipping, parallel redundant synapses, etc. [6, 16, 23, 24, 35, 36, 38, 42, 43].

It may be noted that the works in [6, 24] perform offline training that requires low latency and consume low power as compared to online training. This is because the training algorithm has no direct interaction with the neural network chip, and all computations are carried out by software in the host system. Usually, the remapping and row flipping approaches use online training and hence incur high latency and power overhead as compared to offline training methods [23, 35, 38, 42]. Moreover, the works that use redundant neurons (memristors) further increase the latency, area and power overheads [16, 36, 43].

In this work we use VMM operations for detecting fabrication defects in the crossbar. We carry out analyses to determine the percentage of faults that can be tolerated during the learning phase without incurring additional overheads. Results are provided for both single-layer and multi-layer neural networks for different bit-storage resolutions. A memristor with 1-bit resolution can store two weights, while one with $k$-bit resolution (for $k > 1$) can store $2^k$ distinct weights.

This paper is an extension of the work reported in [40], where the fault tolerance capability of memristor-based single-layer fully-connected neural network has been discussed.

The major contributions of this paper are as follows:

i) The impact of multiple faults on neuromorphic computation on a memristor crossbar has been analyzed.
ii) A detailed study on the effect of faults in both single-layer and multi-layer crossbars has been carried out, with respect to multiple datasets.
iii) An approach for the fault diagnosis of multiple faults in the crossbar has been presented.

The rest of the paper is organized as follows. Section 2 discusses the background and related works. Section 3 analyzes the fault tolerance behavior of the crossbar. Section 4 presents the fault diagnosis framework, while Sect. 5 presents the results in presence of faults on various benchmarks. Finally, Sect. 6 concludes the work.

## 2 Preliminary and Related Works

### 2.1 Memristor

Memristor is the fourth passive electrical element first postulated by Chua in 1971 [7], which shows non-linear relationship between electrical charge and magnetic flux. In 2008, Strukov et al. fabricated a Pt-$TiO_2$-Pt memristor [30]. The $TiO_2$ consists of two regions, a doped region with oxygen vacancies ($TiO_{2-x}$) that is highly conductive, and an undoped region (pure $TiO_2$) that is highly resistive.

The width of the doped region can be changed by applying suitable voltages across the two terminals of the device, which in turn changes the resistive state of the memristor. The resistance value is retained even after voltage is withdrawn, making it a suitable candidate for non-volatile memories [8].

Figure 1 shows the voltage-current (V-I) characteristics of an ideal memristor. The V-I curve forms a pinched hysteresis loop, and according to Chua and Sung [8], any device with this property can be characterized as a memristor. The resistive state of the memristor can be switched between two distinct states by applying voltages with suitable polarity. This property is used in logic applications where the high and low resistive states denote logic 0 and logic 1 respectively.

Another interesting property of memristor is its ability to retain its resistive state even after the power supply is

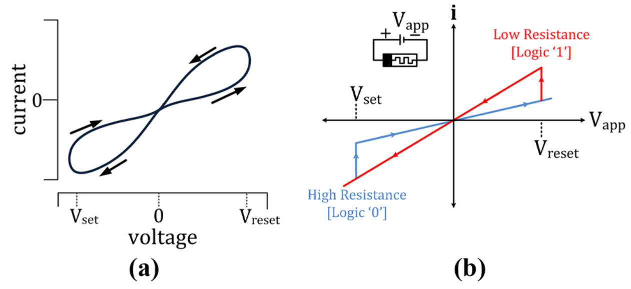


**Fig. 1** Typical V-I characteristic of a memristor: **a** Actual, **b** Ideal

removed. This makes it a viable candidate for use in non-volatile resistive memory (ReRAM).

Typically the voltage $V_{set}$ ($V_{reset}$) is used to expand (shrink) the doped region to make it in low (high) resistive state. Apart from this, several voltage levels can be applied to set the memristors to multiple intermediate resistive states, making it a viable candidate for neuromorphic computing.

## 2.2 Vector-Matrix Multiplication using Memristor Crossbar

Memristors are typically fabricated in a crossbar structure where a device is placed between two perpendicular wires [1]. Such compact placement leads to more storage density as compared to other memory technologies. Such crossbars can also be used to build memory systems that support scalable in-memory computing [20, 33, 39].

In-memory computing is an approach where both storage and computation are performed in the same hardware unit [41]. Figure 2a shows a crossbar structure where we can program the memristors to any desired resistance level. The structure can be used to perform VMM operations as $I = V \times M$ [27, 28, 34]. The voltages applied along the rows can be represented as a vector $V = \{V_0, V_1, \dots, V_{n-1}\}$ and the current flowing along the columns as the vector $I = \{I_0, I_1, \dots, I_{m-1}\}$. The crossbar $M$ can be represented as an $n \times m$ matrix, where the $(i,j)^{th}$ co-efficient denotes the conductance of the memristor in row $R_i$ and column $C_j$.

The dot product operation can be performed in a column of the crossbar as shown in Fig. 2b. Suppose that the conductance value of memristor $M_i$ is $G_i$, for $0 \le i \le (n-1)$. The current flowing through memristor $M_i$ will be $I_i = V_i * G_i$, for $0 \le i \le (n-1)$. The total current flowing in the column will be

$$I_{total} = \sum_{i=0}^{n-1} \left(V_i * G_i\right)$$

which is the dot product of the two vectors $\{V_0, V_1, \dots, V_{n-1}\}$ and $\{G_0, G_1, \dots, G_{n-1}\}$.

Many works have been reported in the literature that uses memristor crossbars to accelerate neuromorphic computations [3, 6, 15, 27, 34].

## 2.3 Fabrication Defects and Fault Models

Circuit defects are anomalies that cause undesired variations between the actual implemented hardware and the anticipated hardware [4, 5]. The defects often occur during the fabrication or operation of the device. The various defects in the crossbar can be characterized as: (i) Imperfect cross-sections – this can cause slow, fast and

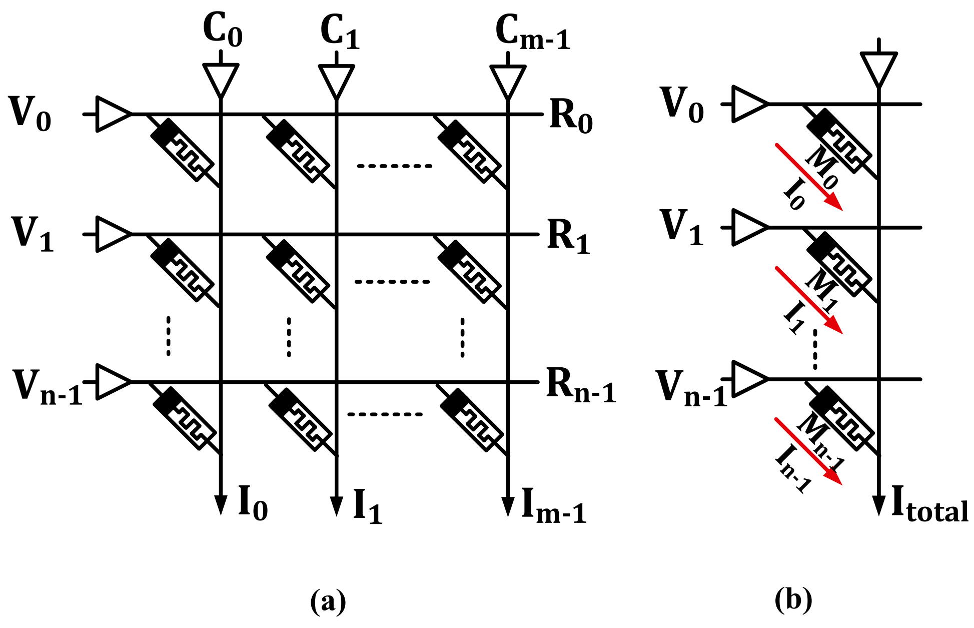


**Fig. 2** **a** A memristor crossbar for VMM operations, **b** Performing dot product in a crossbar column

deep faults; (ii) *Variable oxide thickness* – this can affect the normal resistive switching behavior; (iii) *Open defects* – this can break the crossbar structure thereby causing access issues of a single cell, a row or multiple rows; (iv) *Short defects* – this causes rows/columns to merge, thereby giving rise to coupling faults.

The most general types of faults that may occur can be categorized based on the nature of defects in the crossbar, and are summarized below.

1. *Stuck-At-Fault (SAF):* Here the resistive value of a memristor becomes fixed and cannot be altered (i.e. non-programmable) [5, 17, 18]. Two types of SAF are possible:
   (a) *Stuck-At-Low (SAL):* The memristor is fixed at low conductive state, and arises due to variable oxide thickness.
   (b) *Stuck-At-High (SAH):* The memristor is fixed at high conductive state, and arises due to variable oxide thickness.
2. *Transition Fault (TF):* This is caused due to irregular doping variations, and may prevent a memristor cell from undergoing a resistive state change [5, 17, 18]. Such faults can be classified into two types:
   (a) *Slow-write Fault:* This fault affects the transition time of a device, making it slower than the ideal device. Such a fault can put a memristor in an undesired state during training.
   (b) *Fast-write Fault:* Similarly, this fault causes the transition time to be faster than the ideal memristor.
3. *Address Decoder Fault (ADF/AF):* This fault causes the rows/columns of the crossbar to merge together, thereby causing access to undesired cells in the crossbar. [5, 17, 18].
4. *Deep Fault:* This occurs due to deformity in cross-section and alter the ideal area of the doped region, thereby making the $V_{set}$ ($V_{reset}$) voltage to change the resistance of the memristor to *ON* (*OFF*) state. But upon applying the $V_{reset}$ ($V_{set}$) voltage, the memristor does not switch back to *OFF* (*ON*) state due to the high (low) doped region. This is referred to as Deep-1 (Deep-0) fault [5, 17, 18].
5. *Open Circuit Fault (OCF):* This arises due to open defects, and makes a memristor unaffected by the current flowing in the columns. [5, 17, 18].
6. *Coupling Fault (CF):* This fault occurs due to the coupling of row or column wires. Upon applying a voltage to cause a transition in the affected cell, some other cells adjacent to it may also change state [5, 17, 18].
7. *Read Disturb Fault (RDF):* The RDF is due to the small difference between the $V_{set}$ ($V_{reset}$) and $V_{read}$. This causes wrong read operation, as during reading the cell contents changes its value [5, 17, 18].
8. *Deceptive Read Disturb Fault (DRDF):* This occurs due to oxide variation, making it more sensitive. This fault returns the correct reading value of the target cell but changes the value of the target cell [5, 17, 18].

The architecture of the memristor crossbar and random-access memory (RAM) are similar, due to which a similar fault rate can be anticipated. According to the RAM fault models, the memristor-based crossbar can have ≈30% of faults in it [5]. For instance, experiments with the $HfO_2$ based crossbar show that about 66% of fault-free memristors are available in the crossbar [5]. The major contribution of the faults in the implemented crossbar are: TF (≈14%), SAF (≈10%), ADF (≈3%), RDF and DRDF (≈ 1.5%), and other faults (≈5.5%).

It can be observed that SAF, TF, and DF may arise due to variable oxide thickness and imperfect cross-section [17, 18], and as such share similar characteristics in terms of the defect source. Also, OCF has similar characteristics as SAL, since a memristor with SAL is in a high resistive state.

## 2.4 Related Works on Fault Diagnosis and Fault Tolerance

Though the crossbar structure is similar to traditional RAM, the storage technique is different. Nevertheless, similar fault detection techniques used for conventional RAM also apply for ReRAM. Several techniques available in the literature to mitigate faulty memristors are described as follow:

(a) *March C* Testing*: The authors in [5] proposed an approach to diagnose faulty memristors in the crossbar, which is similar to testing traditional RAM. To diagnose the faulty memristors, the approach selects one row at a time and tests each cell of the row in sequence. To test a single cell the approach requires 6 read and 4 write operations.
(b) *Sneak-path Based Solution*: The authors in [18] proposed a technique to detect faults in the crossbar using sneak-path current. The fault detection is based on the assumption that SAF in memristors generate currents in the anticipated region of the crossbar. This approach is more applicable to the memristor crossbar as compared to March C* based approach, requiring at least 30% fewer cycles.
(c) *Divide-and-Conquer Testing*: The authors in [13] proposed a divide-and-conquer technique to test for SAF. The crossbar is divided into multiple regions, and the individual regions are tested for faults. However, this technique suffers from sneak-path issues. During testing a particular region, the sneak-path current from

some other region may cause the testing of the region to return as faulty.

Several methods have been discussed in the literature to handle performance degradation due to faults [6, 23, 24, 35, 36, 38, 42, 43]. The methods can be broadly classified into three approaches.

i) *Retraining*: The authors in [6, 24] proposed an approach by adjusting the network training, in which low (high) weight is adjusted with SAL (SAH) faulty memristor. The adjustment continues till the crossbar attains the accuracy levels of an ideal array.
ii) *Row Flipping*: The authors in [23, 35, 38, 42] proposed rows interchanging method to minimize the overall error. This process is repeated to minimize the error. In the worst case, the process must be reverted if the solution is undesirable from the previous steps.
iii) *Redundant Neurons*: The authors in [36, 43] proposed a solution by using additional memristors in place of the faulty memristors to handle performance degradation. However, this approach requires complex connections among the memristors and requires initializing the weight in the new additional memristor.

The fault diagnosis approaches available in the literature require significant number of read/write operations, which causes the latency to increase and in turn increases the power consumption. However, this statement is not true for approaches that require offline training, as the neural network chips are not involved during training.

The main objective of this paper is to propose a solution with fewer read/write operations. Due to the architectural advantage of the crossbar, we propose a solution that can perform parallel operations for fault diagnosis.

State-of-the-art methods have reported that in order to overcome the performance degradation caused by faults, 8-37% energy overhead and 9-50% area overhead are incurred [16, 23, 35, 36, 42].

To reduce the overhead required for fault diagnosis, we analyze the tolerance level of faults that will not degrade the performance.

## 3 Fault Tolerance Analysis

There exists several works in the literature [6, 23, 24, 35, 36, 38, 42, 43], that try to handle performance degradation caused by various physical faults. However, these methods require additional operations and hardware overhead, thereby resulting in higher power consumption and delay during both training and inference operations.

It may be noted that for digital operations, even a single fault can cause the outputs to become erroneous. However, neuromorphic computation on a crossbar is essentially an analog operation, where an analog vector is multiplied with a weight matrix. The presence of one or more faults may not always result in erroneous outputs, as the fault effects may get canceled or masked. Since any explicit solution incurs overheads, an analysis is needed to assess the degree of performance degradation. Fault diagnosis approaches for conventional RAMs require large number of read/write operations.

In this work, we have utilized the VMM operations for fault diagnosis in order to reduce the number of time steps and energy consumption.

In the following section, we analyze the percentage of faults in a given crossbar that can be implicitly tolerated, and then propose a fault diagnosis approach.

### 3.1 General Analysis on Effect of SAFs

To illustrate the effect of SAFs in a crossbar, we consider a single column of the crossbar with 4 memristors. We assume that each memristor can store multiple weights as integer values in the range $0-3$, where 0 represents low conductive, 1 and 2 the intermediate, and 3 the high conductive states. We assume that the goal is to achieve output as $\geq 6$ when we apply 1's to all the rows, making the output equal to $\sum 1 \times \mathit{weight\ of\ memristor}$. If the output is more than 6, we can say that the goal is satisfied. We discuss below several cases depending upon the number of faulty memristors and the type of faults.

1. **Case 1:** There are no faults in the crossbar (Fig. 3(case 1)). We can set the weights in $4^4 = 256$ ways out of which 150 will produce desired output ($\geq 6$). This has a probability of $150/256 * 100 = 58.59\%$.
2. **Case 2:** One memristor has a SAL fault (Fig. 3(case 2)). We can set the weights in $4^3 = 64$ ways out of which 20 will produce desired output, with a probability of $20/64 * 100 = 31.25\%$.
3. **Case 3:** Two of the memristors have SAL faults (Fig. 3(case 3)). We can set the weights in 16 ways out of which 1 will produce correct output, and the probability is $1/16 * 100 = 6.25\%$.
4. **Case 4:** Three of the memristors have SAL faults (Fig. 3(case 4)). Here, it is not possible to get the desired output, and the probability will be 0%.
5. **Case 5:** One of the memristors has a SAH fault (Fig. 3(case 5)). We can set the weights in 64 ways out of which 54 will produce desired output, with a probability of $54/64 * 100 = 84.37\%$ (better than ideal case).
6. **Case 6:** Two of the memristors have SAH faults (Fig. 3(case 6)). We can set the weights in 16 ways,

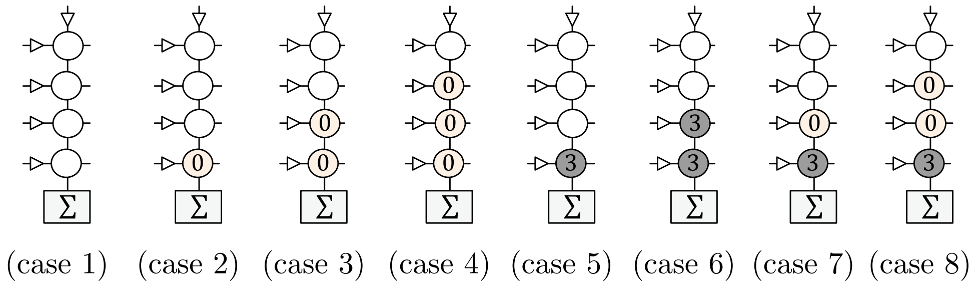


**Fig. 3** Possible faulty cases for four memristors in a column

where all of them will produce desired output. This gives a probability of 100% (better than ideal case).

7. **Case 7:** One memristor has SAL fault, and one has SAH fault (Fig. 3(case 7)). We can set the weights in 16 ways out of which 10 will give desired output, with probability $10/16 * 100 = 62.50\%$ (better than ideal case).
8. **Case 8:** Two memristors have SAL fault, and one has SAH fault (Fig. 3(case 8)). We can set the weights in 4 ways out of which 3 will produce desired output, with a probability of $1/4 * 100 = 25.00\%$. (better than only two SAL fault).

From Fig. 3, it is clear that the occurrence of SAF reduces the programming capability in the crossbar, as some memristors are stuck permanently in the low or high conductive states. Specifically, with a single fault (cases 2 and 5), the programming capability reduces from 256 to 64, and with two faults (cases 3, 6 and 7) it reduces to 16. For this particular example where our objective is to achieve the target as $\geq 6$, we observe that the SAL faults reduce the probability of achieving the desired output (cases 2-4), whereas the SAH faults increase the probability (cases 5-6). However, if our objective is to have an output of $\leq 6$ (i.e., the reverse), then SAL faults will be more beneficial than SAH faults. We can also observe that in the cases where both SAL and SAH faults co-exist (case 7-8), the fault effects get cancelled. This is mainly because the probability improves compared to cases where only SAL exists (cases 2-4), and the probability degrades as compared to cases where only SAH exists (cases 5-6). In general, an in-depth analysis is needed to identify the percentage of faults that can be tolerated.

## 3.2 Analysis of Fault Tolerance Capability of Crossbar

Among the various memristor models available in the literature, some have linear switching characteristics, while some others are non-linear. The switching characteristics of two popular memristor models [21, 26] are shown in Fig. 4.

We can observe that model of [26] switches between two states relatively faster, whereas that of [21] shows slower and linear switching. The linear switching property has the advantage that multiple intermediate resistive states can be set by applying suitable voltage. This is referred to as *multi-bit storage resolution* [12, 14, 25, 31]. For instance, 1-bit resolution can have $2^1 = 2$ resistive states, 2-bit resolution can have $2^2 = 4$ states, and so on. Clearly, an increase in the storage resolution provides greater programming flexibility and higher accuracy but also increases the hardware complexity. In this paper, we consider devices with up to 5-bit resolution, which translates into $2^5 = 32$ resistive states.

To carry out experimentation on fault tolerance, we have chosen the following datasets: *MNIST*, *Extended-MNIST*, *Fashion MNIST*, *HAR* and *Cifar-10* [2, 9, 11, 19, 22, 37]. We have used direct downloading method [32], where the training is done offline, and the computed weights are

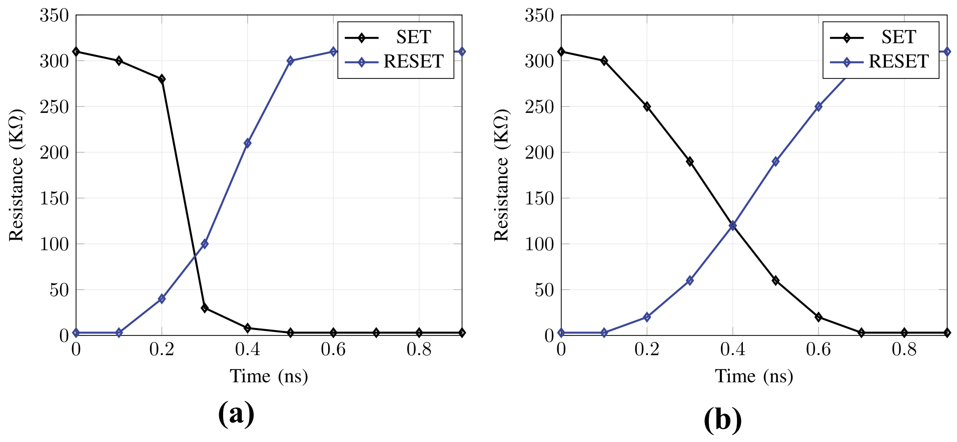


**Fig. 4** Switching characteristics in: **a** Stanford model [26], **b** VTEAM model [21]

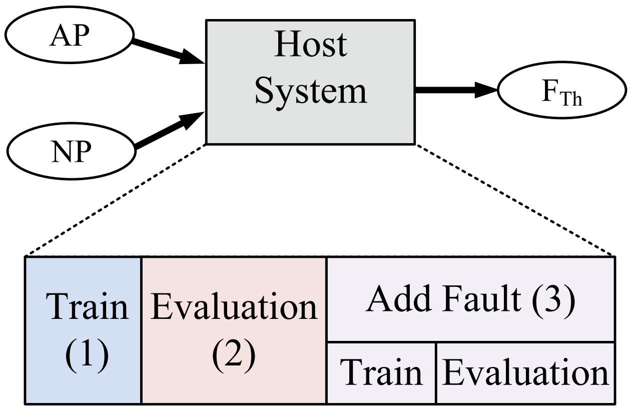


**Fig. 5** Framework for fault tolerance analysis (AP: Application/dataset, NP: Network parameter, F$_{Th}$: Fault tolerance threshold

downloaded to the crossbar for testing. The framework for analysis is depicted in Fig. 5.

As shown in Fig. 5, the application dataset (AP) and the network parameters (NP) are applied as inputs to the *Host System (HS)*. NP consists of various crossbar characteristics like bit-capacity, size of crossbar, $R_{on}$ to $R_{off}$ ratio, etc. HS will perform three tasks:

a) *Training:* The application is trained by performing all necessary computations and the weights generated.
b) *Evaluation:* With the generated weights, evaluation is performed to find the accuracy of the network for an ideal crossbar.
c) *Add Faults:* By varying the percentage of injected faults, training and evaluation are carried out again, and the evaluated accuracy is compared against the ideal case. If the accuracy lies within $\pm 1$, the number of faults is increased and step (c) is repeated. If the accuracy of faulty crossbar reduces by more than 1%, then the fault percentage is returned as *Fault Tolerance Threshold* ($F_{th}$) for the application.

The key advantage of off-line training is that we can use any suitable algorithm for training. However, in off-line training, small errors in the given parameters can create large differences in the accuracy values. Thus, a verification step is needed for host-based off-line training. The off-line training and analysis framework has been implemented in Python and run on an Intel i7 based desktop with 2.6 GHz clock and 8GB RAM running Ubuntu. We have used the back-propagation algorithm to train the applications, and modified it to meet the required constraints (viz. non-negative weights, the weights of SAF memristors should not be modified during weight change, etc.). We have used the Stanford memristor model [26] to simulate the crossbar, which has been carried out under Cadence Virtuoso environment.

All the chosen datasets are trained using fully-connected neural networks, with the size of crossbar being $N_a \times N_c$, where $N_a$ and $N_c$ respectively denote the number of attributes and number of classes. For example, to train with the dataset for MNIST handwritten digits, $784 \times 10$ crossbar is used on which VMM operations are performed.

The main objective of this paper is to analyse the effect of fault, where the analysis is mostly carried out on a single layer crossbar. We have extended it for multi-layer crossbar as well; however the hardware connection across layers is beyond the scope of this paper.

Figure 6 shows training loss of MNIST data set for up to 5-bit resolution for an ideal crossbar. We observe that the training loss reduces with increase in the storage resolution. Table 1 summarizes the percentage test accuracy for different datasets with respect to various storage resolutions.

From Table 1 we observe that with every bit increase in resolution, the accuracy increases by $\approx 1\%$ for all datasets. However, the training time and hardware complexity will increase for higher resolutions.

For evaluating the fault tolerance capability, faults are injected in the crossbar and training is performed again in order to analyze the effect of faults. Figure 7 shows variation in training loss for various % of faults using MNIST dataset with 1-bit resolution.

From the figure we observe that the variation in training loss is similar to that of an ideal crossbar for up to 4% of faults. In other words, the overall accuracy is not affected by up to 4% of faults. In order to speed up the analysis, we carry out testing by injecting faults in random positions in the crossbar, and repeat the process multiple times to average out the statistical variations.

Figure 8 shows variation in test accuracy for 1% faults with 1-bit resolution for the MNIST dataset. For some test run,

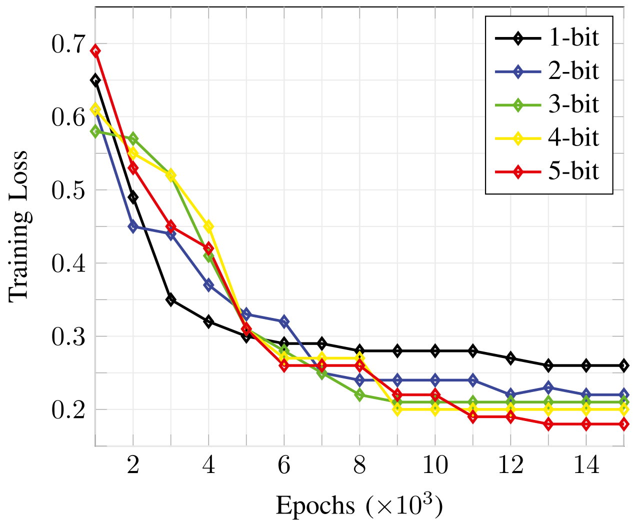


**Fig. 6** Training loss for MNIST dataset

**Table 1** Percentage accuracy for ideal crossbars

| No. of Bits | MNIST [22] | EMNIST [9] | HAR [2] | CIFAR-10 [19] | FMNIST Bits [37] |
|---|---|---|---|---|---|
| 1-bit | 74.06 | 67.32 | 82.33 | 70.18 | 71.45 |
| 2-bit | 77.92 | 72.41 | 87.02 | 72.75 | 73.29 |
| 3-bit | 79.06 | 72.93 | 88.92 | 74.29 | 74.78 |
| 4-bit | 80.23 | 74.64 | 90.52 | 75.83 | 75.45 |
| 5-bit | 81.40 | 75.28 | 91.15 | 76.54 | 76.80 |

presence of fault improves the performance of the network, whereas for some cases it decreases the performance. Also the average variation in test accuracy is $\pm 1$. For some cases, the test accuracy reduces by more than 5% as compared to ideal case. After detailed analysis we found that this is caused because most of the faults ($\geq 40\%$) are mapped to the same crossbar column. This results in classification error for the class corresponding to that column, thereby reducing the overall accuracy. However, the proposed fault diagnosis approach can identify such column(s), and we can avoid mapping of any class to such column(s).

Figure 9 shows the average variation in test accuracy for the MNIST dataset for different storage resolutions.

In the Fig 9, the y-axis shows variation in accuracy with respect to the ideal case (as represented by line 0). If the accuracy of the faulty crossbar reduces by more than 1 (lies outside the green lines), then the fault percentage is returned as *Fault Tolerance Threshold* ($F_{Th}$) for the application.

It can be observed that for resolution of 1 or 2, the variation in test accuracy with number of faults is not significant. However, the faulty memristors can affect crossbars with higher resolution to a greater extent. From Fig. 9 we can conclude that up to 4% of the faults can be tolerated if we are considering 1-bit resolution. However, for higher resolutions the figure drops to 3% for the MNIST dataset. Similar behavior have been observed for other datasets as well.

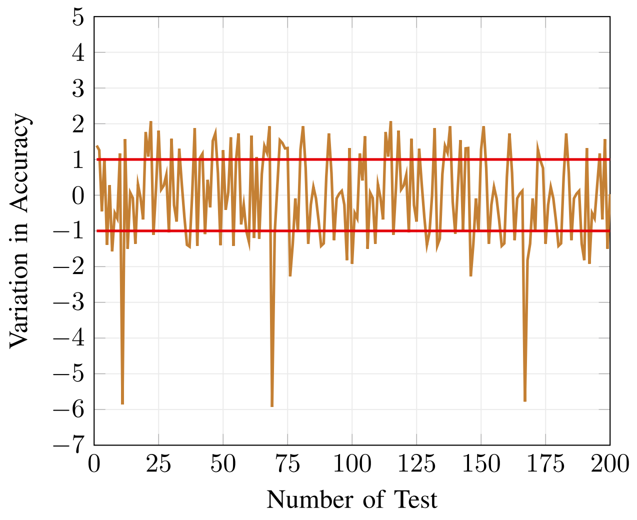


**Fig. 8** Variation in test accuracy for MNIST dataset with 1-bit resolution. The x-axis (Number of Test) indicates the number of analyses the dataset undergoes with random fault positions

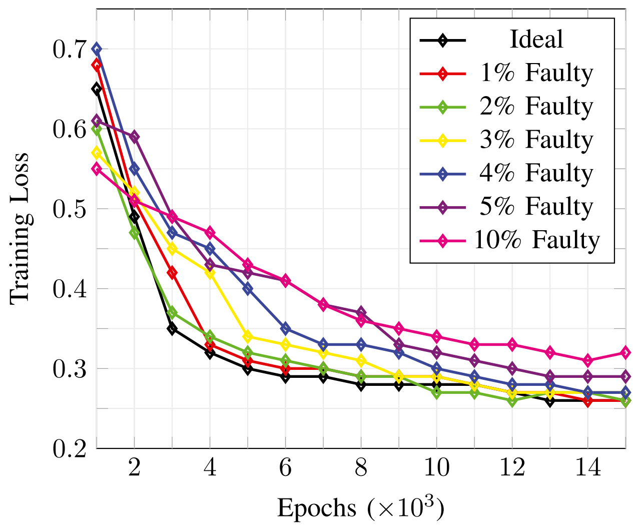


**Fig. 7** Training loss for MNIST dataset (Faulty Crossbar)

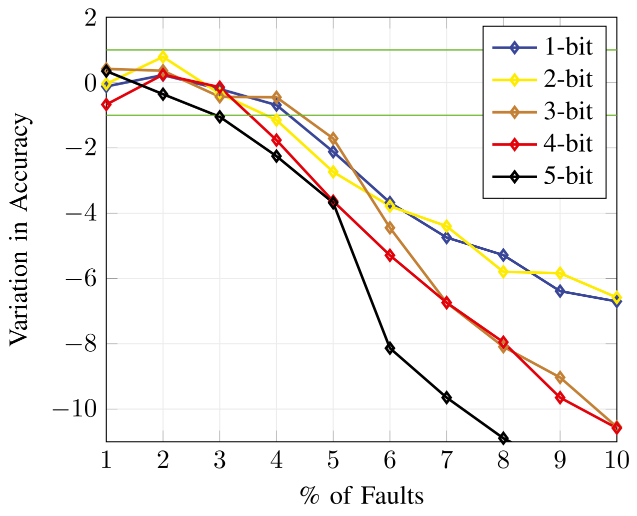


**Fig. 9** Average variation in accuracy with faults for MNIST dataset in multi-bit crossbar

**Table 2** Percentage of faults that can be tolerated

| No. of Bits | MNIST [22] | EMNIST [9] | HAR [2] | CIFAR-10 [19] | FMNIST [37] |
|---|---|---|---|---|---|
| 1-bit | ≤4 | ≤4 | ≤5 | ≤3 | ≤3 |
| 2-bit | ≤4 | ≤3 | ≤4 | ≤3 | ≤3 |
| 3-bit | ≤3 | ≤3 | ≤4 | ≤2 | ≤2 |
| 4-bit | ≤3 | ≤2 | ≤3 | ≤2 | ≤2 |
| 5-bit | ≤2 | ≤1 | ≤2 | ≤1 | ≤2 |

Based on the average variations from single to multi-bit resolutions, the threshold value for fault tolerance for various datasets have been calculated and depicted in Table 2.

From the table we observe that for any dataset implemented with 1-bit resolution there may not be any performance degradation if the memristors in the crossbar are $\leq 4\%$ faulty. For 2-bit resolution, it will be $\leq 3\%$; and for more than 2-bit resolution, it reduces to $\leq 2\%$. It is obvious that the faults will not be in exact ratio, and hence we have performed testing by considering different ratios of SAL/SAH faults.

We use the notation $K1$:$K2$, where $K1$ and $K2$ respectively denote the proportion of SAFs for low conductance and high conductance respectively. Through experimental analyses, it has been observed that the threshold characteristics of 1:10 and 10:1 are very similar to 1:1.

### 3.3 Fault Tolerance Analysis for Multi-layer and Other Faults

For many large and complex datasets, we have to use multi-layer neural networks for better performance. With this motivation, we have also analyzed the effect of faults in multi-layer neural networks.

Table 3 shows the percentage accuracy and threshold values generated by neural networks with 1 and 2 hidden layers respectively for MNIST dataset, with various storage resolutions. The threshold values shown in the table is the average of fault tolerance capabilities across all the layers. However, it has been observed that the last layer of the crossbar is more sensitive to faults. This is because in the last layer, any error in the crossbar can directly affect the classification accuracy.

We have analyzed the effects of slow and fast write faults for higher resolution crossbars. This is done by applying suitable voltage to switch the memristor state to the next conductance level, and then reading the state and comparing it with that of an ideal memristor. Experimental results show that the fault tolerance capability does not change much for slow/fast write faults.

**Table 3** Percentage accuracy and threshold value for multi-layer networks

| No. of Bits | Percentage Accuracy | | Percentage Threshold | |
|---|---|---|---|---|
| | 1 layer | 2 layers | 1 layer | 2 layers |
| 1-bit | 80.60 | 82.45 | ≤5 | ≤5 |
| 2-bit | 84.52 | 87.12 | ≤4 | ≤4 |
| 3-bit | 85.24 | 88.85 | ≤3 | ≤4 |
| 4-bit | 86.54 | 91.26 | ≤2 | ≤3 |
| 5-bit | 88.32 | 92.74 | ≤2 | ≤2 |

## 4 Fault Diagnosis Framework

Existing approaches to fault diagnosis in the crossbar are either based on conventional approaches used for RAMs, or rely on additional circuitry. In this work, we have used crossbar VMM operations directly to identify different types of faults, which incurs less power consumption and number of cycles. We proceed in two steps. In the first step of *high-level fault detection*, we only identify the percentage of faults and not their locations. For higher percentage of faults that may cause performance issues, the exact locations of the faults are identified in the second step, referred to as *low-level fault detection*. We consider stuck-at faults in the crossbar for fault diagnosis.

Figure 10 shows the overall approach for fault diagnosis. We first carry out high-level fault detection to identify the percentage of faults (say, $F_{perc}$). If $F_{perc} \leq F_{Th}$, where $F_{Th}$ denotes the fault tolerance threshold for the application, the weights as learnt are loaded to the crossbar. This significantly reduces the energy/delay overheads.

However, if $F_{perc} > F_{Th}$, we use the low-level fault detection approach instead of the conventional fault diagnosis

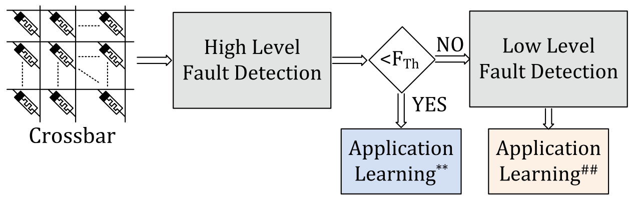


**Fig. 10** Framework for fault diagnosis

algorithms [5, 13, 18]. Subsequently, learning can be performed using retraining/remapping algorithms [6, 23, 35, 36, 42].

```
Algorithm 1 Algorithm for High-Level Fault Detection
    INPUT: $N \times M$ Crossbar
    OUTPUT: $F_{perc}$
1: Detect SAH faults
2:   RESET Crossbar
3:   Perform VMM
4:   Calculate number of SAH memristors
5: Detect SAL faults
6:   SET Crossbar
7:   Perform VMM
8:   Calculate number of SAL memristors
9: Calculate % of SAH
```

i. *RESET* Crossbar: Apply reset voltage $V_{reset}$ to all rows. With this all programmable memristors will be set with low conductance value. But, memristors that are stuck on high conductance state will not change state.
ii. *SET* Crossbar: Apply set voltage $V_{set}$ to all rows. With this all programmable memristors will be set with high conductance value. But, memristors that are stuck on low conductance state will not change state.
iii. Perform VMM: Apply read voltage $V_{read}$ to each row. With this vector of read voltages will be multiplied with conductance values of the memristor crossbar, and the overall current is measured in respective columns.
iv. Calculate number of SAL/high memristors: Compare the actual current injected during VMM operation and compute the number of high conductance memristors.
v. Calculate % of SAH: Based on number of SAL and SAH memristors, compute % of SAH.

## 4.1 High-Level Fault Detection

The classification accuracy of a neuromorphic system depends on the weights as computed during the learning phase. However, in the presence of faults some of the cells may be rendered non-programmable, thereby resulting in degradation of accuracy. However, as discussed in the previous section, this may not always be true, as the effects of some set of multiple faults may cancel each other.

The value of $F_{perc}$ can be determined directly using VMM operation. First we set (or reset) all the memristors in the crossbar, where the faulty memristors will not change their states. By applying read voltage ($V_{read}$) along the rows and measuring the currents in the columns, we can estimate the value of $F_{perc}$

The currents in the crossbar columns after performing VMM operation is given by $V \times G$ (vide Fig. 2b). For an $N \times M$ crossbar, in the fault-free case the total column currents after reset operation will be:

$$I_{R_{read}} = N * V_{read} * G_{off} \tag{1}$$

where $G_{off}$ is the low-conductance value of a memristor.

Similarly, total column currents after set operation will be:

$$I_{S_{read}} = N * V_{read} * G_{on} \tag{2}$$

**Table 4** Parameters for simulation

| Parameter | $X_{on}$ | $X_{off}$ | $W_{on}$ | $W_{off}$ | $R_{on}$ | $R_{off}$ |
|---|---|---|---|---|---|---|
| Value | 9 nm | 0 nm | 5 nm | 0.5 nm | 3 KΩ | 1.66 MΩ |

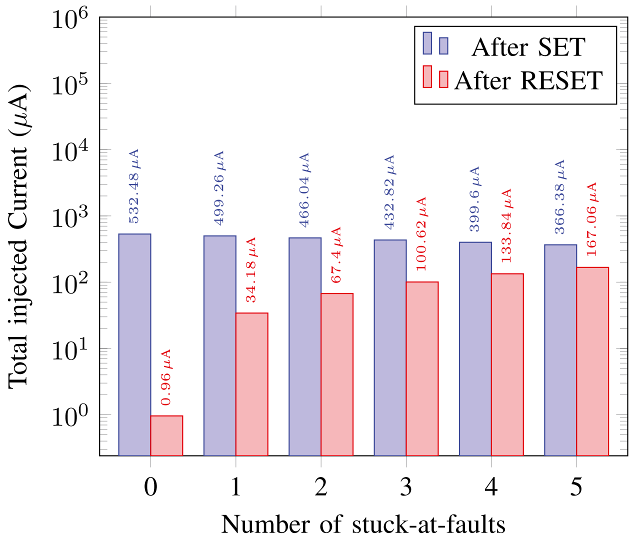


**Fig. 11** Total injected current in different columns consist various number of SAFs after Set/Reset operation

where $G_{on}$ denotes the high-conductance value of a memristor.

In general, if the number of high-conductance and low-conductance memristors in any column are $x$ and $y$ respectively, the current injected in the column will be:

$$I_{actual} = \left(x * V_{read} * G_{on}\right) + \left(y * V_{read} * G_{off}\right) \tag{3}$$

Since $G_{on} >> G_{off}$, we can write $I_{actual} \approx A * V_{read} * G_{on}$. The number of SAH and SAL memristors in the column can be calculated as:

$$N_{sah} = I_{R_{read}} / (V_{read} * G_{on}) \tag{4}$$

$$N_{sal} = N - I_{S_{read}} / (V_{read} * G_{on}) \tag{5}$$

Using Eqs. (4) and (5) we can calculate $V_{perc}$. This finding has been verified through simulation under the Cadence Virtuoso environment for a 16 × 16 crossbar, where we have used the Stanford memristor model [26] with the parameters given as in Table 4 and read voltage of 100mV.

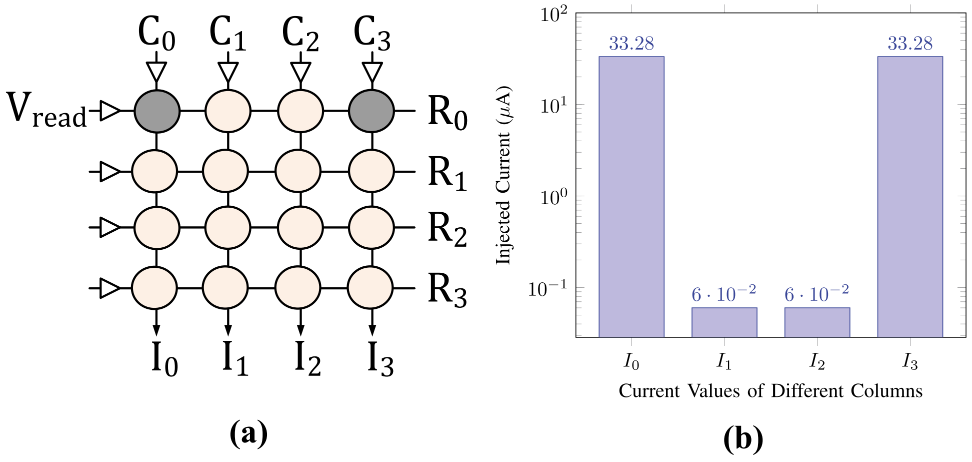
Fig. 11 shows the currents in two different columns of a 16 × 16 crossbar with varying number of faulty memristors. The currents injected is approximately proportional to the product of number of high-conductance memristors, $V_{read}$ and $G_{on}$. The currents generated by low-conductance and high-conductance memristors are $0.06\mu A$ and $33.28\mu A$ respectively. The blue bars represent the currents injected by $V_{read}$ after set operation, whereas orange bars represent the currents injected by $V_{read}$ after reset operation. From the figure we can observe that after set operation, the total current injected is $532.48\mu A$ and $499.26\mu A$ for fault-free and single SAH fault cases respectively. This may be verified as follows:

(a) The current $532.48\mu A$ is actually the sum of those generated by 16 high-conductance memristors (i.e. $16 * 33.28\mu A$);
(b) The current $499.26\mu A$ is the sum of those generated by 15 high-conductance and 1 low-conductance memristors (i.e. $\approx 15 * 33.28 + 0.06$);

and so on.

For small crossbars, we may ignore the currents generated by SAL conductance memristors; however, for large crossbars the overall error contribution can be significant. For instance, 555 low-conductance memristors in a column will contribute to current $555 * 0.06 = 33.3\mu A$, which becomes comparable to the current for a high-conductance memristor.

**Fig. 12** Current variations due to SAH memristors



Similar calculations can be carried out for the detection of stuck-at high conductance memristors. Consider the orange bars that indicate the currents injected by $V_{read}$ after reset operation. For the fault-free case it will generate $0.96\mu A$ current, whereas a single SAH memristor will cause a high current of $34.18\mu A$.

**Algorithm 2** Algorithm for Low-Level Fault Detection

```
    INPUT: $N \times M$ Crossbar
    OUTPUT: SAH memristors index
 1: Detect SAH memristor positions
 2: RESET Crossbar
 3: for $Row_{i=0}$ to $N$-1 do
 4:     $Row_i = V_{read}$
 5:     Measure Current and Locate
 6:     High Fault Index = $(Row_i, Column_{high})$
 7: end for
 8: Detect SAL memristor positions
 9: SET Crossbar
10: for $Row_{i=0}$ to $N$-1 do
11:     $Row_i = V_{read}$
12:     Measure Current and Locate
13:     Low Fault Index = $(Row_i, Column_{low})$
14: end for
15: Return Low Fault and High Fault Index
```

i. RESET Crossbar: Similar to Algorithm 1.
ii. SET Crossbar: Similar to Algorithm 1.
iii. Measure Current and Locate: Read current from all columns in parallel and store into High/Low Fault Index.
iv. High/Low Fault Index: During SAH test store every index $(Row_i, Column_{high})$. Similarly, during SAL test store every index $(Row_i, Column_{low})$.
v. $Row_i$: Row where $V_{read}$ voltage have been applied.
vi. $Column_{high}$: All columns from $Column_0$ to $Column_{M-1}$ generating high current.
vii. $Column_{low}$: All columns from $Column_0$ to $Column_{M-1}$ generating low current.

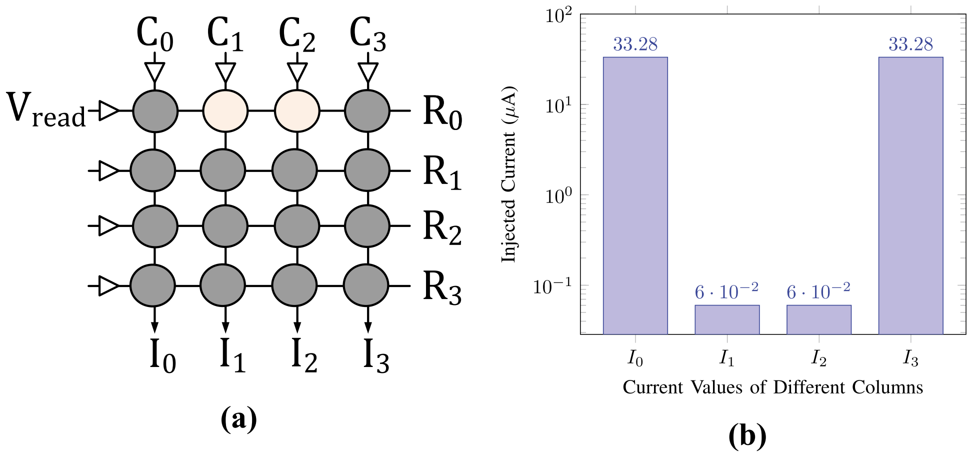


**Fig. 13** Current variations due to SAL memristors

**Algorithm 3** Algorithm for DRDF Fault Detection

```
    INPUT: $N \times M$ Crossbar
    OUTPUT: DRDF memristors index
 1: SET Crossbar
 2: for $Row_{i=0}$ to $N$-1 do
 3:     $Row_i = V_{read}$
 4:     Measure Current and Store in Buffer
 5:     $Row_i = V_{read}$
 6:     DRDF Fault Index = $(Row_i, Col_{cur} \neq Buffer_i)$
 7: end for
 8: RESET Crossbar
 9: for $Row_{i=0}$ to $N$-1 do
10:     $Row_i = V_{read}$
11:     Measure Current and Store in Buffer
12:     $Row_i = V_{read}$
13:     DRDF Fault Index = $(Row_i, Col_{cur} \neq Buffer_i)$
14: end for
15: Return DRDF Fault Index
```

i. DRDF Fault Index: Store every index of $Row_i$ for which the current generated by $V_{read}$ is $\neq$ to current value stored in Buffer.
ii. All other parameters are similar to that of Algorithm 1 and 2.

## 4.2 Low-Level Fault Detection: Fault Diagnosis

For $F_{perc} > F_{Th}$, we need to identify the exact locations of the faulty memristors in the crossbar. All the faulty memristors in a row can be determined in a single cycle.

To identify the SAH conductance memristors, we reset the crossbar, apply read voltage to one row(say $R_i$) at a time, and measure the currents in each column. If a column (say $C_j$) shows high current, it means that the memristor located at the junction $(R_i, C_j)$ is faulty. A current comparator circuit as proposed in [10] can be used for the purpose (see Fig. 12). As can be seen from the plot, the currents generated in the faulty columns (33$\mu$A) are much greater as compared to the fault-free columns (0.06$\mu$A).

The same process can be repeated, where we set the crossbar, and identify columns with low currents for detecting SAL memristors as shown in Fig. 13. Here, the currents generated in SAL faulty columns (0.06$\mu$A) are very less as compared to those in non-faulty columns (33.26$\mu$A).

For an $N \times M$ crossbar, the process to identify all the faulty memristors will take 2$N$ time steps, where $N$ time steps will be required to detect SAH memristors, and $N$ time steps for SAL memristors. The steps are formally stated in Algorithm 4.2.

**Algorithm 4** Algorithm for Slow/Fast Write Fault Detection.

```
   INPUT: $N \times M$ Crossbar
   OUTPUT: Slow/Fast write memristors index
1: RESET Crossbar
2: for $Row_i = 0$ to $N$-1 do
3:     for State = 1 to n do
4:         $Row_i = V_{sc}$
5:         Measure Current and Locate
6:         Fast Write Index = $(Row_i, Column_{low})$
7:         Slow Write Index = $(Row_i, Column_{high})$
8:     end for
9: end for
```

i. $V_{sc}$: A voltage value which is sufficient to increase the state of memristor by 1.
v All other parameters are similar to that of Algorithm 2.

## 4.3 Multiple Fault Diagnosis with VMM

In the previous subsections, we have discussed the high-level and low-level fault diagnosis techniques with respect to stuck-at faults in the memristors. However, the VMM operations that are used for fault diagnosis can also detect other types of faults as explained below.

1. *Deceptive Read Destructive Fault (DRDF):* To identify these faults in the crossbar, we require two read operations instead of one. The first read operation is used to

**Table 5** Analysis of the variability in resistance with one SAF memristor

| $F_T$ | $I_i$ ($\mu$A) | | $Var_p$ | % Var | $I_{var}$ ($\mu$A) | | $\|I_{var} - I_i\|$ ($\mu$A) | |
|---|---|---|---|---|---|---|---|---|
| | $I_{i_{set}}$ | $I_{i_{reset}}$ | | | $I_{var_{set}}$ | $I_{var_{set}}$ | $I'_{set}$ | $I'_{reset}$ |
| 1 SAF with 1 Variable Memristor | | | | | | | | |
| | | | | -20 | 506.67 | – | 7.41 | – |
| | | | | -20 | 506.67 | – | 7.41 | – |
| | | | | -15 | 504.53 | – | 5.27 | – |
| | | | | -10 | 502.51 | – | 3.25 | – |
| SAL | 499.26 | – | $R_{on}$ | -5 | 501.71 | – | 2.45 | – |
| | | | | +5 | 498.25 | – | 1.01 | – |
| | | | | +10 | 495.55 | – | 3.71 | – |
| | | | | + 15 | 494.48 | – | 4.78 | – |
| | | | | + 20 | 494.27 | – | 4.99 | – |
| | | | | -20 | – | 34.41 | – | 0.23 |
| | | | | -15 | – | 34.31 | – | 0.13 |
| | | | | -10 | – | 34.28 | – | 0.10 |
| | | | | -5 | – | 34.22 | – | 0.04 |
| SAH | – | 34.18 | $R_{off}$ | +5 | 498.25 | – | 1.01 | – |
| | | | | +10 | 495.55 | – | 3.71 | – |
| | | | | + 15 | 494.48 | – | 4.78 | – |
| | | | | + 20 | 494.27 | – | 4.99 | – |
| 1 SAF with 2 Variable Memristors (Both memristors showing same type of variability on $R_{on}/R_{off}$) | | | | | | | | |
| | | | | -5,-20 | 509.93 | – | 10.67 | – |
| | | | | -5,-15 | 507.04 | – | 7.78 | – |
| | | | | -5,-10 | 505.29 | – | 6.03 | – |
| SAL | 499.26 | – | $R_{on}$ | +5,+10 | 495.12 | – | 4.14 | – |
| | | | | +5,+15 | 493.91 | – | 5.35 | – |
| | | | | +5,+20 | 492.66 | – | 6.60 | – |
| | | | | -5,-20 | – | 34.42 | – | 0.24 |
| | | | | -5,-15 | – | 34.37 | – | 0.19 |
| | | | | -5, -10 | – | 34.35 | – | 0.17 |
| SAH | – | 34.18 | $R_{off}$ | +5,+10 | – | 33.34 | – | 0.84 |
| | | | | +5,+15 | – | 33.20 | – | 0.98 |
| | | | | +5,+20 | – | 33.17 | – | 1.01 |
| | | | | +5,+15 | – | 33.20 | – | 0.98 |
| | | | | +5,+20 | – | 33.17 | – | 1.01 |
| 1 SAF with 2 Variable Memristors (Both memristors showing opposite variability on $R_{on}/R_{off}$) | | | | | | | | |
| | | | | -5,+5 | 499.26 | – | 0.00 | – |
| SAL | 499.26 | – | $R_{on}$ | -5,+10 | 498.29 | – | 0.97 | – |
| | | | | -5,+15 | 497.09 | – | 2.17 | – |
| | | | | -5,+20 | 495.98 | – | 3.28 | – |
| SAH | – | 34.18 | $R_{off}$ | -5,+5 | – | 34.18 | – | 0.00 |
| | | | | -5,+10 | – | 34.22 | – | 0.04 |
| | | | | -5,+15 | – | 34.21 | – | 0.03 |
| | | | | -5,+20 | – | 34.19 | – | 0.01 |

sensitize all DRDF faults present in a row, while the second read operation confirms the presence or absence of the faults. If $I_{ij,1}$ and $I_{ij,2}$ denote the currents through the memristor at position $(i, j)$ in the crossbar after the first and second read operations respectively, then the memristor will have DRDF fault if $I_{ij,1} \neq I_{ij,2}$. The process can be integrated with stuck-at fault detection as shown in Algorithm 1, which leads to a 2× increase in complexity as compared to Algorithm 4.2.

2. *Slow/Fast Write Fault:* For a memristor with 1-bit resolution, the conductance value that can be stored is two-valued. The voltages $V_{set}$ and $V_{reset}$ are applied across a memristor to change its conductance value to high and low levels, respectively. The presence of slow/fast write faults in the crossbar does not affect the performance for 1-bit resolution devices; however, for $k$-bit resolution ($k > 1$) there can be some performance degradation. To diagnose the slow/fast write affected memristive cells, we apply a voltage that is sufficient to change the state of memristors by +1 in a row-by-row manner, and verify whether read operation is successful. If a memristor shows different current value than the others, it indicates that it is suffering from slow/fast write fault. The process is summarized in Algorithm 4.3. If we integrate this approach with Algorithm 4.2, the complexity increases O($2^k$)×, where $k$ is the storage resolution of the memristors.

Memristor-based crossbars are prone to various kind of failures; however, all of them cannot be detected using VMM operations alone. Also some faults such as RDF that occur due to the small differences between $V_{set}$ ($V_{reset}$) and $V_{read}$ can be minimized by making the difference larger. Other faults such as coupling faults result in state change in pairs of adjacent cells, and are bidirectional in nature. Detection of the faulty cells in this case cannot be done using VMM operation, and would require writing of certain patterns on the adjacent memory cells that can excite the faulty behavior.

## 4.4 Effect of Process Variation

It may be noted that the memristors that constitute the crossbar are prone to process variations that can in turn affect the process of diagnosis and identification of faulty memristors. One of the common types of process variation is *device-to-device variation*, which describes the variability in the switching behaviour of the memristor cells. In this section, we analyze the effect of variations in $R_{on}$ and $R_{off}$ values (up to ±20%) in the proposed high-level fault detection approach.

Table 5 shows the currents injected in a column of a $16 \times 16$ crossbar, with one cell having SAF and one or more cells having variations in $R_{on}/R_{off}$ values. The first column ($F_T$) shows the type of fault, while the second column shows the current ($I_i$) flowing in the $i^{th}$ column. It is assumed that one of the memristors in column $i$ is faulty, while all others have ideal $R_{on}$ and $R_{off}$ values. The second column consists of two parts which represent the currents $I_{i_{set}}$ and $I_{i_{reset}}$ injected in the column $i$ by applying $V_{read}$ after Set and Reset operations, respectively. The third column (Var$_p$) shows which of $R_{on}$ or $R_{off}$ has variations. The fourth column (%Var) denotes the % resistance variation ranging from −20 to +20 %. The fifth column ($I_{var}$) shows the injected current by applying $V_{read}$ in the $i^{th}$ column of the crossbar considering a SAF memristor along with memristor(s) showing variability. This column consists of two parts as $I_{var_{set}}$ and $I_{var_{reset}}$ which represents read after the Set and Reset operations respectively. The last column shows the difference between the ideal and the actual current.

For the case with one SAF memristor and no variations, the total currents injected after Set and Reset operations are 499.26$\mu$A and 34.18$\mu$A (see Fig. 11). For analysis, we compare this case with the currents injected after Set and Reset operations, considering one or more memristors having resistive variability.

From Table 5 we observe that if we have a few memristors that show variability in their resistance, then the difference between $I_i$ and $I_{var}$ is very small, and will not cause any error in high-level fault detection. The error arises if $|I_{var} - I_i|$ is greater than the current injected by a single high conductance memristor (say, $I_{read}$).

From Fig. 11 we can observe that, each fault affects the overall injected current by $\approx 33\,\mu A = I_{read}$.

For instance, the bar representing one SAF shows total injected current 499.26$\mu$A (34.18$\mu$A) by applying $V_{read}$ after Set (Reset) operations, indicating that there is one SAL (SAH) faulty memristor available in that column. Now consider that due to variation, if the current injected by $V_{read}$ after set operation becomes $\geq 532\,\mu A \approx 16 \times I_{read}$ indicating the approach will report zero SAL memristor instead of one. Similarly, if the current injected by $V_{read}$ after reset operation becomes $\geq 67\,\mu A \approx 2 \times I_{read}$ indicating that the approach will report two SAHs memristor instead of one.

Thus, if $|I_{var} - I_i|$ is larger than $I_{read}$, then the approach will generate an error of ±1. Similarly, if $|I_{var} - I_i|$ is larger than $n\times I_{read}$, then we can have an error of $\pm n$. This can be generalized as:

$$E_{SAL} = |I_{var_{set}} - I_{i_{set}}| / I_{read} \tag{6}$$

$$E_{SAH} = |I_{var_{reset}} - I_{i_{reset}}| / I_{read} \tag{7}$$

**Table 6** Comparative study for $N \times M$ crossbar

| Parameter | Fault Diagnosis Approach | | | |
|---|---|---|---|---|
| | In [13] | In [5] | In [18] | Proposed |
| Read Operation | 2N + N log(N) | 3N | 2N | 2N |
| Write Operation | 2N | 4N | 4N | 2N |
| Read Cycle | 2 + N log(N) | 3N | 2N | 2n |
| Write Cycle | 2 | 4N | 4N | 2 |

We also observe that a small variation in $R_{on}$ increases the overall current by a substantial amount, but the same is not true for small variations in $R_{off}$.

In our study, we have analysed the currents for a column with one SAF and up to two variable memristors considering ±20% resistive variability. A detailed analysis is required to understand the effects of process variations in multiple memristors, which can be taken up as future work.

## 5 Results and Discussion

In this section, we compare the read/write operations and latency required by the proposed work with respect to existing methods in the literature.

Let $T_{set}$, $T_{reset}$, and $T_{read}$ denote the times required to set, reset and read a memristor in the crossbar respectively. For a $n \times m$ crossbar which consist of $N$ memristors, the authors in [5] has shown that $2N$ Reset, $2N$ Set, and $6N$ Read operations are required to diagnose multiple faults in the crossbar. Though their main purpose was to detect various types of coupling faults, detection of a single cell stuck-at fault requires 2 Reset, 2 Set, and 3 Read operations. Similarly, in the work reported in [18], diagnosis of multiple faults require $2N$ Reset, $2N$ Set, and $4N$ Read operations, and single cell SAFs with 2 Reset, 2 Set, and 2 Read operations. The diagnosis approach proposed in [5, 18] dose not support parallel execution of operations and thus requires one cycle for each operation.

In [13], the authors proposed a method where the faulty columns are identified first followed by a divide-and-conquer approach to diagnose the SAL faults first, and then the SAH faults. Although the number of cycles required has not been reported in the work, we can make a rough estimation as follows.

- Before detection of faulty memristors firstly the crossbar is Set/Reset, which requires total of $N$ Set and $N$ Reset operations. However, all memristors can be Set or Reset in a single cycle; thus we need 2 cycles for this (1 cycle for Set and 1 cycle for Reset).
- After Set/Reset operation, read operation is performed to identify faulty columns. The $N$ read operations required can be done in parallel, and thus requires 2 cycles (1 cycle to read after Set, and 1 cycle to read after Reset).
- Based on the region of the fault, crossbar is split using divide-and-conquer approach to identify locations of faulty memristors; this is performed twice (once to identfy SAL and once for SAH memristors). This requires $2N \log N$ read operations that can be done with $2N \log N$ cycles.

The method proposed in this paper requires 1 Reset, 1 Set, and 2 Read operations to detect single cell SAF, and 1 Reset, 1 Set, and 2N Read operations for detecting SAFs in $N$ memristors. However, as all the memristors present in a row can be diagnosed in single cycle, we require $2 + 2n$ cycles to perform these operations.

In general, the Set and Reset operations are considered as write operation.

Table 6 compares the read/write operations, and also the corresponding number of cycles, of the proposed method as compared to the works reported in [5, 13, 18].

The table shows that the proposed approach requires 50% less write operations as compared to [5, 18], and ≈30% less read operations as compared to [5, 13]. In our approach, a crossbar write can be done in 2 cycles, and read in $2n$ cycles for an $n \times m$ crossbar. The proposed work requires $O(n)$ cycles for fault detection, whereas [5, 13, 18] requires $O((nm)^2)$ cycles. However, the proposed approach incurs the additional overhead of one comparator in each column.

If we assume that read and write cycles requires same amount of time (say, 1 unit), then latency for [5, 13] and [18] will be $\approx N \log N$ ((nm) log(nm)), $\approx 7N$ (7nm) and $\approx 6N$ (6nm) respectively. Whereas, latency for the proposed work will be $\approx 2n$.

The methods proposed in [16, 23, 35, 36, 42] requires about 20-50% more computation to compensate performance degradation caused by faults, and incur 8-37% energy and 9-50% area overhead. In contrast, the proposed approach trains the crossbar directly if it finds that $F_{perc} \leq F_{Th}$, thereby reducing power consumption and time overheads.

## 6 Conclusion

Resistive RAM crossbars have drawn the attention of researchers for neuromorphic computing due to their capability of low-power VMM operation. SAFs in the crossbar can affect the programmability of the memristors and degrade the accuracy of VMM operations. For handling performance degradation, the state-of-the-art solutions require additional computation and circuitry and hence consume more power and delay. In the crossbar, it is possible that the effects of several faults may cancel each other, thus leading to fault tolerance capability. The proposed work analyses the percentage of such faults that can be tolerated. We have proposed a fault diagnosis approach that analyzes the VMM operation to diagnose the faulty crossbar cells, which have linear time complexity in terms of the number of rows. Thus overall reduction in delay and power can be achieved using this approach.

**Author Contributions** All authors have equally contributed in the study and writing of this manuscript. The manuscript has been read and approved for submission by all authors

**Data Availability Statement** This publication is supported by multiple datasets, which are openly available at locations cited in the reference section. Documentation for MNIST dataset is available at [22], and is openly available for download at http://yann.lecun.com/exdb/mnist/. Documentation for EMNIST dataset is available at [9], and is openly available for download at https://www.westernsydney.edu.au/icns/reproducible_research/publication_support_materials/emnist. Documentation for HAR dataset is available at [2], and is openly available for downloaded at https://archive.ics.uci.edu/ml/machine-learning-databases/00240/. Documentation of CIFAR-10 data is available at [19], and is openly available for downloaded at https://www.cs.toronto.edu/~kriz/cifar.html. Documentation of FMNIST is available at [37], and is openly available for download at https://github.com/zalandoresearch/fashion-mnist

## Declarations

**Ethical Approval** Not applicable.

**Consent to Participate** Not applicable.

**Conflicts of Interest** The authors declare no conflict of interest.

## References

1. Akarvardar K, Wong HSP (2009) Ultralow voltage crossbar nonvolatile memory based on energy-reversible NEM switches. IEEE Electron Device Lett 30(6):626–628. https://doi.org/10.1109/LED.2009.2018289
2. Anguita D, Ghio A, Oneto L, Parra X, Reyes-Ortiz JL, etal. (2013) A public domain dataset for human activity recognition using smartphones. In: Proc. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning ESANN 3
3. Ankit A, Hajj IE, Chalamalasetti SR, Ndu G, Foltin M, Williams RS, Faraboschi P, Hwu WmW, Strachan JP, Roy K, Milojicic DS (2019) PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In: Proc. 24th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems 715-731. https://doi.org/10.1145/3297858.3304049
4. Bushnell ML, Agrawal VD (2004) Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits. vol 17, Springer Science and Business Media, chap 4
5. Chen CY, Shih HC, Wu CW, Lin CH, Chiu PF, Sheu SS, Chen FT (2015) Rram defect modeling and failure analysis based on march test and a novel squeeze-search scheme. IEEE Trans Comput 64(1):180–190. https://doi.org/10.1109/TC.2014.12
6. Chen L, Li J, Chen Y, Deng Q, Shen J, Liang X, Jiang L (2017) Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar. In: Proc. Design, Automation Test in Europe Conference Exhibition (DATE)19-24. https://doi.org/10.23919/DATE.2017.7926952
7. Chua LO (1971) Memristor-the missing circuit element. IEEE Transactions on Circuit Theory 18(5):507–519. https://doi.org/10.1109/TCT.1971.1083337
8. Chua LO, Sung MK (1976) Memristive devices and systems. Proc IEEE 64(2):209–223. https://doi.org/10.1109/PROC.1976.10092
9. Cohen G, Afshar S, Tapson J, van Schaik A (2017) EMNIST: Extending MNIST to handwritten letters. In: Proc. International Joint Conference on Neural Networks (IJCNN) IEEE 2921–2926. https://doi.org/10.1109/IJCNN.2017.7966217
10. Danaboina YKY, Samanta P, Datta K, Chakrabarti I, Sengupta I (2019) Design and implementation of threshold logic functions using memristors. In: Proc. 32nd International Conference on VLSI Design and 18th International Conference on Embedded Systems (VLSID) 518–519. https://doi.org/10.1109/VLSID.2019.00115
11. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
12. Gu P, Li B, Tang T, Yu S, Cao Y, Wang Y, Yang H (2015) Technological exploration of rram crossbar array for matrix-vector multiplication. In: Proc. The 20th Asia and South Pacific Design Automation Conference 106–111. https://doi.org/10.1109/ASP-DAC.2015.7058989
13. Hongal VA, Kotikalapudi R, Kim YB, Choi M (2011) A novel "divide and conquer" testing technique for memristor based lookup table. In: Proc. IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS) 1-4. https://doi.org/10.1109/MWSCAS.2011.6026406
14. Hossam H, Mamdouh G, Hussein H, El-Dessouky M, Mostafa H (2018) A new read circuit for multi-bit memristor-based memories based on time to digital sensing circuit. In: Proc. IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS) 1114–1117. https://doi.org/10.1109/MWSCAS.2018.8623907
15. Hu M, Graves CE, Li C, Li Y, Ge N, Montgomery E, Davila N, Jiang H, Williams RS, Yang JJ, Xia Q, Strachan JP (2018) Memristor-based analog computation and neural network classification with a dot product engine. Adv Mater 30(9):1705914. https://doi.org/10.1002/adma.201705914
16. Huangfu W, Xia L, Cheng M, Yin X, Tang T, Li B, Chakrabarty K, Xie Y, Wang Y, Yang H (2017) Computation-oriented fault-tolerance schemes for rram computing systems. In: Proc. 22nd Asia and South Pacific Design Automation Conference (ASP-DAC) 794–799. https://doi.org/10.1109/ASPDAC.2017.7858421
17. Kannan S, Rajendran J, Karri R, Sinanoglu O (2013) Sneak-path testing of memristor-based memories. In: Proc. 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems. 386–391. https://doi.org/10.1109/VLSID.2013.219
18. Kannan S, Karimi N, Karri R, Sinanoglu O (2015) Modeling, detection, and diagnosis of faults in multilevel memristor memories. IEEE Trans Comput Aided Des Integr Circuits Syst 34(5):822–834. https://doi.org/10.1109/TCAD.2015.2394434
19. Krizhevsky A, Nair V, Hinton G (2009) Cifar-10 (canadian institute for advanced research) 5:4. http://www.cs.toronto.edu/kriz/cifar.html
20. Kvatinsky S, Belousov D, Liman S, Satat G, Wald N, Friedman EG, Kolodny A, Weiser UC (2014) Magic–memristor-aided logic. IEEE Transactions on Circuits and Systems II: Express Briefs 61(11):895–899. https://doi.org/10.1109/TCSII.2014.2357292
21. Kvatinsky S, Ramadan M, Friedman EG, Kolodny A (2015) VTEAM: A general model for voltage-controlled memristors.

IEEE Trans Circuits Syst II Express Briefs 62(8):786–790. https://doi.org/10.1109/TCSII.2015.2433536
22. LeCun Y (1998) The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/
23. Liu B, Li H, Chen Y, Li X, Wu Q, Huang T (2015) Vortex: Variation-aware training for memristor x-bar. In: Proc. 52nd ACM/EDAC/IEEE Design Automation Conference (DAC) 1–6. https://doi.org/10.1145/2744769.2744930
24. Liu M, Xia L, Wang Y, Chakrabarty K (2019) Fault tolerance in neuromorphic computing systems. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference 216-223 https://doi.org/10.1145/3287624.3288743
25. Merced-Grafals EJ, Dávila N, Ge N, Williams RS, Strachan JP (2016) Repeatable, accurate, and high speed multi-level programming of memristor 1T1R arrays for power efficient analog computing applications. Nanotechnology 27(36):365202. https://doi.org/10.1088/0957-4484/27/36/365202
26. Reuben J, Fey D, Wenger C (2019) A modeling methodology for resistive RAM based on Stanford-PKU model with extended multilevel capability. IEEE Trans Nanotechnol 18:647–656. https://doi.org/10.1109/TNANO.2019.2922838
27. Shafiee A, Nag A, Muralimanohar N, Balasubramonian R, Strachan JP, Hu M, Williams RS, Srikumar V (2016) Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In: Proc. ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA) 14–26. https://doi.org/10.1109/ISCA.2016.12
28. Sheridan PM, Cai F, Du C, Ma W, Zhang Z, Lu WD (2017) Sparse coding with memristor networks. Nat Nanotech (12):784-789. https://doi.org/10.1038/nnano.2017.83
29. Song L, Qian X, Li H, Chen Y (2017) Pipelayer: A pipelined ReRAM-based accelerator for deep learning. In: Proc. IEEE International Symposium on High Performance Computer Architecture (HPCA) 541–552 https://doi.org/10.1109/HPCA.2017.55
30. Strukov DB, Snider GS, Stewart DR, Williams RS (2008) The missing memristor found. Nature 453(7191):80–83. https://doi.org/10.1038/nature06932
31. Taherinejad N, Manoj PS, Jantsch A (2015) Memristors' potential for multi-bit storage and pattern learning. In: Proc. IEEE European Modelling Symposium (EMS) 450–455. https://doi.org/10.1109/EMS.2015.73
32. Tam S, Gupta B, Castro H, Holler M (1990) Learning on an analog VLSI neural network chip. In: Proc. IEEE International Conference on Systems, Man, and Cybernetics Conference Proceedings 701–703. https://doi.org/10.1109/ICSMC.1990.142209
33. Thangkhiew PL, Gharpinde R, Datta K (2018) Efficient mapping of boolean functions to memristor crossbar using magic nor gates. IEEE Trans Circuits Syst I Regul Pap 65(99):2466–2476. https://doi.org/10.1109/TCSI.2018.2792474
34. Xia L, Gu P, Li B, Tang T, Yin X, Huangfu W, Yu S, Cao Y, Wang Y, Yang H (2016) Technological exploration of RRAM crossbar array for matrix-vector multiplication. J Comput Sci Technol 31(1):3–19. https://doi.org/10.1007/s11390-016-1608-8
35. Xia L, Liu M, Ning X, Chakrabarty K, Wang Y (2017) Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems. In: Proc. 54th ACM/EDAC/IEEE Design Automation Conference (DAC) 1–6. https://doi.org/10.1145/3061639.3062248
36. Xia L, Huangfu W, Tang T, Yin X, Chakrabarty K, Xie Y, Wang Y, Yang H (2018) Stuck-at fault tolerance in RRAM computing systems. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 8(1):102–115. https://doi.org/10.1109/JETCAS.2017.2776980
37. Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms http://arxiv.org/abs/1708.07747
38. Xu Q, Chen S, Geng H, Yuan B, Yu B, Wu F, Huang Z (2020) Fault tolerance in memristive crossbar-based neuromorphic computing systems. Integration 70:70–79. https://doi.org/10.1016/j.vlsi.2019.09.008
39. Yadav DN, Thangkhiew PL, Datta K (2019) Look-ahead mapping of boolean functions in memristive crossbar array. Integration 64:152–162. https://doi.org/10.1016/j.vlsi.2018.10.001
40. Yadav DN, Datta K, Sengupta I (2020) Analyzing fault tolerance behaviour in memristor-based crossbar for neuromorphic applications. In: 2020 IEEE International Test Conference India 1–9 https://doi.org/10.1109/ITCIndia49857.2020.9171788
41. Yang JJ, Strukov DB, Stewart DR (2013) Memristive devices for computing. Nature nanotechnology 8(1):13. https://doi.org/10.1038/nnano.2012.240
42. Zhang B, Uysal N, Fan D, Ewetz R (2020a) Handling stuck-at-fault defects using matrix transformation for robust inference of dnns. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39(10):2448–2460. https://doi.org/10.1109/TCAD.2019.2944582
43. Zhang B, Uysal N, Fan D, Ewetz R (2020b) Redundant neurons and shared redundant synapses for robust memristor-based DNNs with reduced overhead. In: Proceedings of the ACM Great Lakes Symposium on VLSI 339-344. https://doi.org/10.1145/3386263.3406910

**Dev Narayan Yadav** received the M.Tech. degree in Computer Science and Engineering from National Institute of Technology Meghalaya, India, in 2018. He worked as a Junior Research Fellow at the National Institute of Technology Meghalaya during 2018 to 2019. He is currently pursuing his PhD degree from the Indian Institute of Technology Kharagpur, India. His research interests include memristor and its applications in logic design and neuromorphic computing.

**Phrangboklang Lyngton Thangkhiew** received the Bachelor of Engineering (B.E.) from Visvesvaraya Technological University, India, in 2014, and PhD from National Institute of Technology Meghalaya, India, in 2019. He worked as an Assistant Professor at Vellore Institute of Technology Chennai, India, during 2020 to 2021 and as a Post-Doctoral Fellow at Indian Institute of Technology Guwahati, India, from October, 2021 to January, 2022. He is currently working as an Assistant Professor at the Indian Institute of Information Technology Guwahati, India. He has published several papers in peer-reviewed journals and conferences. His research interests include memristor and its applications in logic design, synthesis, and design optimization.

**Kamalika Datta** completed her Master of Science (MS) from Indian Institute of Technology Kharagpur, India in 2010, and Ph.D. from Indian Institute of Engineering Science and Technology (IIEST), Shibpur, India in 2014. She worked as Assistant Professor at the National Institute of Technology Meghalaya from August 2014 to 2018, Post-Doctoral Research Fellow at the Nanyang Technological University Singapore from February, 2019 to September, 2020, and as an Associate Professor at JIS University Kolkata from November 2020 to August 2021. She is presently working as a Senior Researcher at the Deutsches Forschungszentrum fürKünstliche Intelligenz (DFKI) Bremen, Germany from October 2021. She has published more than 75 papers in peer reviewed journals and conferences. Her research interests include logic design using emerging technologies, synthesis and optimization of reversible and quantum circuit, and embedded systems.

**Sandip Chakraborty** received the B.E. degree from Jadavpur University, Kolkata, India, and the M.Tech. and PhD degrees from the Indian Institute of Technology Guwahati, India. He is currently an Associate

Professor at the Indian Institute of Technology Kharagpur, India. He is an Associate Editor of the Ad-Hoc Networks Journal (Elsevier) and Pervasive and Mobile Computing Journal (Elsevier). He has authored more than 100 papers in peer-reviewed journals and conferences. His current research interests include applications of machine learning over computer systems and networks, broadly in network protocol design and performance optimization, network measurement, and sensing systems.

**Rolf Drechsler** received the Diploma and Dr. Phil. Nat. degrees in computer science from J.W. Goethe University Frankfurt am Main, Frankfurt am Main, Germany, in 1992 and 1995, respectively. He was with the Institute of Computer Science, Albert-Ludwigs University, Freiburg im Breisgau, Germany, from 1995 to 2000, and with the Corporate Technology Department, Siemens AG, Munich, Germany, from 2000 to 2001. Since October 2001, he has been with the University of Bremen, Bremen, Germany, where he is currently a Full Professor and the Head of the Group for Computer Architecture, Institute of Computer Science. In 2011, he additionally became the Director of the Cyber-Physical Systems group at the German Research Center for Artificial Intelligence (DFKI) in Bremen. His current research interests include the development and design of data structures and algorithms with a focus on circuit and system design. Rolf Drechsler was a member of Program Committees of numerous conferences including, e.g., DAC, ICCAD, DATE, ASP-DAC, FDL, MEMOCODE, and FMCAD, and Symposium Chair ISMVL 1999 and 2014, Symposium Chair ETS 2018, and the Topic Chair for “Formal Verification” DATE 2004, DATE 2005, DAC 2010, as well as DAC 2011. He received best paper awards at HVC in 2006, FDL in 2007 and 2010, DDECS in 2010, and ICCAD in 2013 and 2018. He is an Associate Editor of TCAD and JETC. He is an ACM Distinguished Member and an IEEE Fellow.

**Indranil Sengupta** received his B.Tech., M.Tech., and PhD degrees in Computer Science from the University of Calcutta in 1983, 1985, and 1990 respectively. He joined the Indian Institute of Technology Kharagpur, India, as a faculty member in 1988, in the Department of Computer Science and Engineering. Currently, he is the Vice-Chancellor of JIS University, Kolkata, India. He had been the former Heads of the Department of Computer Science and Engineering and the School of Information Technology. He has over 32 years of teaching and research experience, guided 22 PhD students, and published over 200 papers in peer-reviewed journals and conferences. He has served as the Program Chair and General Chair for several International Conferences in the areas of VLSI design/test, reversible computing, and information security. His research interests include reversible and quantum computing, VLSI design and test, and network security. He is a Senior Member of the IEEE.