2024 IEEE International Conference on Design, Test and Technology of Integrated Systems (DTTIS) | 979-8-3503-6312-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/DTTIS62212.2024.10779989

Fault Tolerant Design for Memristor-based AI Accelerators

Theofilos Spyrou, Artemis Zografou, Said Hamdioui and Anteneh Gebregiorgis
Computer Engineering Lab, Delft University of Technology, Delft, The Netherlands
Email: {T.Spyrou, S.Hamdioui, A.B.Gebregiorgis}@tudelft.nl, A.Zografou@student.tudelft.nl

***Abstract*—Computation-In-Memory (CIM) using emerging memristive devices offers a promising solution to implementing energy efficient Artificial Intelligence (AI) hardware accelerators. Though, the non-idealities characterizing memristive devices cause a negative impact on the performance of CIM-based micro-architectures. We propose a two-step fault tolerance strategy to address the impact of Stack-at Faults (SAFs) and conductance variation of RRAM crossbar arrays, composed of a fault tolerant activation function and a retraining method. Evaluation results on Binary Neural Network (BNNs) architectures trained with MNIST, Fashion-MNIST, and CIFAR-10 datasets demonstrate that the proposed techniques can restore the classification accuracy by up to 20%, 40% and 80%, respectively.**

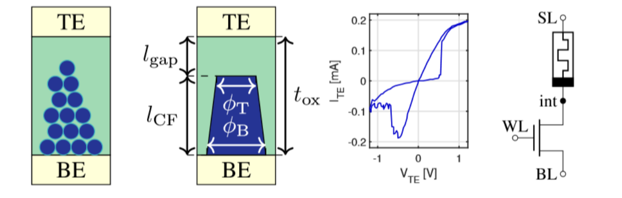
***Index Terms*—CIM, RRAM, Fault Tolerance, Reliability, BNNs**

I. Introduction

Computation-In-Memory (CIM) is a computing paradigm that integrates computation and storage at the same physical location [1]. Implemented with emerging memristive non-volatile devices, such as Resistive Random Access Memory (RRAM), memristor-based CIM architectures can circumvent the costly data movement of Von-Neumann systems, thus, improving significantly energy efficiency and latency while offering superior performance [2]. The structure of CIM crossbar arrays makes them ideal for Multiply & Accumulate (MAC) operations, which are fundamental in Artificial Intelligence (AI) algorithms, like Deep Neural Networks (DNNs).

Albeit the numerous advantages of memristor-based CIM architectures, among which zero leakage, non-volatility, high density, etc., their practical implementation often suffers from non-idealities and manufacturing defects, such as conductance drift, read disturb, and Stack-At Faults (SAFs) [3]. Hence, to ensure a reliable employment of memristive devices, addressing these issues is of utmost importance.

In the effort to mitigating the impact of memristor non-idealities, several software-based [4], [5], [6], [7] and hardware-based [8], [9] fault tolerance mechanisms have been proposed. Software-based solutions focus on finding an optimal mapping of the weights on the memristive crossbar in order to accommodate for SAFs [4], [5]. Another technique is to construct a failure map indicating the locations of the defective devices and then use it to retrain the network on-chip, so that it can learn around [6], [7]. In hardware-based solutions on the other hand, it is common to rely on redundancy schemes to improve fault tolerance. In [8], a re-configurable crossbar array is proposed using redundant columns, while the work in [9] proposes a modified four-transistor-one-resistor (4T1R) cell structure.



(a) Ox. vacancies (b) Tech. parameters (c) I-V curve (d) 1T-1R cell

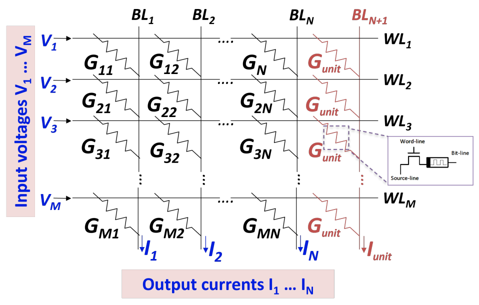
Fig. 1: The structure and operation of a RRAM device [10].

In this work we propose an efficient two-step fault tolerance strategy to address the impact of SAFs and conductance variation on the performance of Binary Neural Networks (BNNs) mapped to RRAM-based CIM AI hardware accelerators. An impact analysis is performed first to evaluate the sensitivity of different BNN architectures to SAFs and conductance variation. Then, the first part of the fault tolerance strategy investigates the role of different activation functions on suppressing this impact and selects a fault tolerant activation function to help restore the classification accuracy. Finally, a retraining method is applied to accommodate for the performance degradation caused by the defective RRAM devices.

The rest of the article is structured as follows. Sec. II introduces the principles of RRAM-based CIM. Sec. III presents the defect modelling of RRAM devices, which is then used in Sec. IV to perform an impact analysis. Sec. V presents the proposed fault tolerance techniques to minimize the impact of RRAM defects and showcases the derived results. Finally, Sec. VI concludes the paper.

II. CIM with RRAM crossbar arrays

The structure and operation of a RRAM device is shown in Fig. 1. As shown in Fig. 1a, a RRAM has a metallic oxide layer, e.g., HfO_x, or TiO_x, sandwiched between Top Electrode (TE) and Bottom Electrode (BE) [10]. To program the device, a sufficiently high voltage is applied on the two electrodes, which, depending on the polarity, forms or dissolves the Conductive Filament (CF) (blue dots in Fig. 1a) and the device is set to the Low or High Resistance State (LRS, HRS), respectively. Furthermore, RRAM devices can be set to intermediate resistance states as well, i.e., multi-level operation, by applying appropriate write pulses. Then, to read the device's state, a small voltage is applied and the current through the



BL_1
BL_2
BL_N
BL_{N+1}
V_1
V_2
V_3
V_M
WL_1
WL_2
WL_3
WL_M
G_{11} G_{12} G_N G_{unit}
G_{21} G_{22} G_{2N} G_{unit}
G_{31} G_{32} G_{3N} G_{unit}
G_{M1} G_{M2} G_{MN} G_{unit}
I_1 I_2 I_N I_{unit}
Input voltages V_1 ... V_M
Output currents I_1 ... I_N
Word-line
Bit-line
Source-line

Fig. 2: The RRAM crossbar array.

device is sensed [11]. The applied read voltage needs to be small enough in order not to alter the state of the cell.

RRAM devices are non-volatile and can support high integration density, which makes them appealing to a plethora of applications, such as non-volatile memories [12], programmable logic [13], and neuromorphic computing [14]. To efficiently design RRAM-based CIM architectures, cells are stacked in a $M \times N$ crossbar array, as shown in Fig. 2, where M and N are the numbers of wordlines and bitlines, respectively.

The input voltages $V_1, V_2, ..., V_M$ are applied on the corresponding wordlines and are shared by all cells at the same row. The current I_{ij} passing through cell c_{ij}, that connects row i to column j, is the product of the input voltage V_i and the cell's conductivity G_{ij}, or $I_{ij} = V_i G_{ij}$. According to Kirchhoff's law, the total output current emerging bitline j is equal to the sum of all cell currents across the column, or $I_j = \sum_i V_i G_{ij}$. At the end of each bitline, an Analog-to-Digital Converter (ADC) or a Sense Amplifier (SE) converts the resulted currents to digital values.

This operation is equivalent to the dot product of the input voltage vector $\boldsymbol{V}$ and the conductance vector $\boldsymbol{G_j}$ at column j, or $I_j = \boldsymbol{V} \times \boldsymbol{G_j}$. Extending to the whole array, N dot products are calculated simultaneously, performing a Matrix-Vector Multiplication (MVM) in a single step. When used with a DNN, each column of the crossbar array holds the synaptic weights of a neuron, which are then multiplied with the inputs.

Given a BNN, the synaptic weight $W_{ij} \in \{-1, 1\}$ needs to be mapped to a conductance value $G_{ij} \in \{0, 1\}$, as we use a 1T1R cell structure [15] that we operate in two states, i.e., LRS (logic '1') or HRS (logic '0'). To do so, the mapping function of Eq. 1 is employed and the total current I_j at the end of bitline j is given by Eq. 2:

$$W_{ij} = 2\, G_{ij} - 1 \tag{1}$$

$$I_j = \sum_i^M W_{ij} V_i = 2 \sum_i^M G_{ij} V_i - \sum_i^M V_i \tag{2}$$

where $\sum_i^M V_i$ is common for all columns and is calculated once by an added extra unit column with conductance values G_{unit} all set to LRS [16], as shown in Fig. 2 with red color.

III. Defect modeling of RRAM devices

Despite the key role of memristors in enabling energy-efficient and highly-parallel CIM architectures, the non-idealities characterizing them, hinder their actual operation. Same applies for RRAM devices, which frequently suffer from various reliability and variability problems. These problems are classified in two main categories, namely, the *time-zero* and *time-dependent* non-idealities [3].

Time zero refers to the moment of fabrication of the device and includes non-idealities such as conductance variation, wire parasitics, and over-forming, caused by fabrication imperfections and the stochastic nature of the underlying physics. Time-dependent non-idealities regard the limited endurance of these devices, degradation due to stress and ageing, conductance drift, and read disturb, causing the stored value to change state. Most of these defects present a transient effect on the operation of RRAM devices, while some others can cause a permanent damage. For example, the accumulated effect of large numbers of read/write operations leading to significant conductance drift [17], or the over-forming caused by excessive process variations, make the defective RRAM devices to get stack at LRS (stack-at-1, SA1) or HRS (stack-at-0, SA0).

In this work, we focus on the impact of SAFs and conductance variation. First, SA0 and SA1 faults are injected randomly into the crossbar array and their impact is evaluated. In case that the original weight value matches the one of the SAF, then the injected fault has no effect. Moreover, the impact of conductance variation is assessed next, which can aggregate the impact and characteristics of other non-idealities, such as conductance drift, read disturb etc. Conductance variation is modeled as a normal distribution ($\mu \pm 3\sigma$) of the conductance of RRAM devices.

TABLE I: The BNN architectures and their performance.

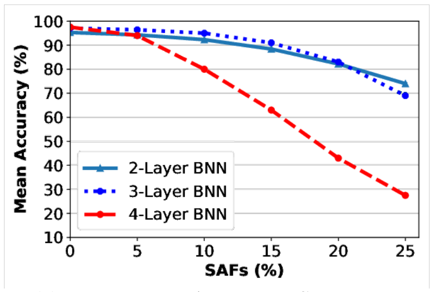
Architecture	Dataset	Accuracy (%)
n-layer FC BNN, $n \in \{2, 3, 4\}$	MNIST	$95.3, 97.1, 97.8$
4-layer FC BNN	Fashion-MNIST	88.6
VGG BNN	CIFAR-10	90.1

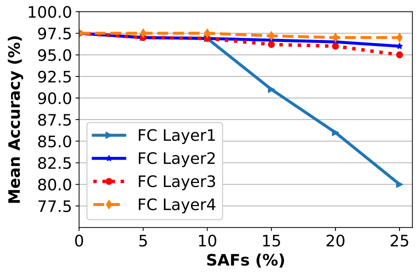
IV. RRAM defects impact analysis

For a corroborated analysis, we examine three case studies based on the MNIST, Fashion-MNIST, and CIFAR-10 datasets. First, a Fully Connected (FC) BNN architecture varying from 2 to 4 layers with 784 neurons in each hidden layer is trained on the MNIST dataset. The same 4-layer BNN is also used with the Fashion-MNIST dataset. The VGG BNN, composed of 6 convolutional layers with 3 max-pooling layers in-between them and 3 fully connected layers at the end, is trained on the CIFAR-10 dataset. Table I summarizes the selected case studies and their baseline performance.

A. Stuck-at fault impact

First, we analyze the impact of SAFs on the 2-, 3-, and 4-layer BNN architectures and the results are presented in Fig. 3a. As shown, the deeper architectures tend to be more sensitive, mainly, due to the increased number of neurons, which results in a larger absolute number of simultaneously injected faults.


(a) Impact on n-layer MNIST BNNs.


(b) Layer-wise impact on MNIST BNN.

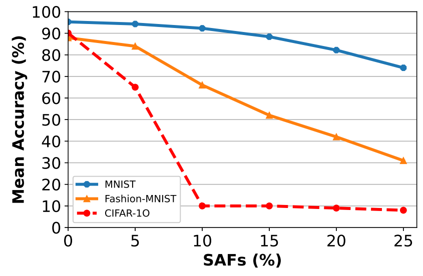
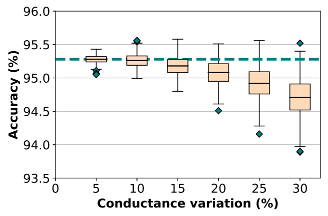

(c) Impact on different BNN architectures.

Fig. 3: SAF impact evaluated for different (a) architecture depth, (b) layers, and (c) architectures.


(a) 2-layer MNIST BNN

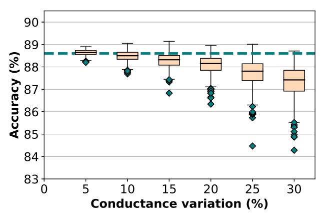

(b) Fashion-MNIST BNN

Fig. 4: Conductance variation impact.

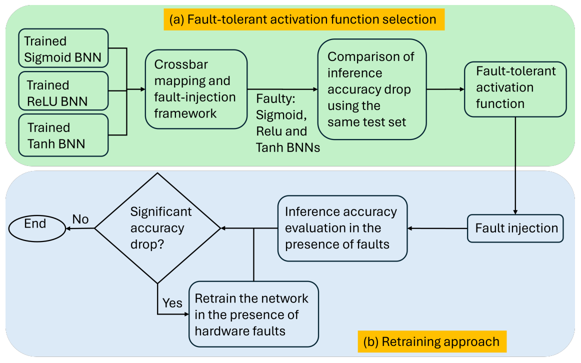

(a) Fault-tolerant activation function selection
Trained Sigmoid BNN
Trained ReLU BNN
Trained Tanh BNN
Crossbar mapping and fault-injection framework
Faulty: Sigmoid, Relu and Tanh BNNs
Comparison of inference accuracy drop using the same test set
Fault-tolerant activation function
Fault injection
Inference accuracy evaluation in the presence of faults
Significant accuracy drop?
No
End
Yes
Retrain the network in the presence of hardware faults
(b) Retraining approach

Fig. 5: The fault tolerance flow for a trained BNN.

Next, we assess the impact of SAFs layer-wise for the 4-layer BNN architecture. Fig. 3b shows the mean accuracy drop when SAFs are injected to each layer of the network, targeting one layer at a time. We observe that the impact is larger for the first layer, while being less severe for the upcoming layers. The reason behind this is mainly the fact that the effects of a fault occurring early in the network are propagated to the rest of the layers and thus are magnified. Additionally, the output layer consists of a relatively small number of neurons (10), therefore the absolute number of injected faults is very small.

Finally, Fig. 3c presents the impact of SAFs on networks trained with different datasets. The results agree with the previous ones, showcasing that the shallower architectures trained on the MNIST dataset are less sensitive compared to the deeper ones required by the Fashion-MNIST and CIFAR-10 datasets.

B. *Conductance variation impact*

Regarding conductance variation, it inserts a minor impact of no more than 5% drop of the classification accuracy, even for large variations up to 30% of the nominal conductance values, as shown in Figs. 4a and 4b for the 2-layer MNIST BNN and the Fashion-MNIST BNN, respectively. Therefore, conductance variation has a significantly less severe impact on the classification accuracy compared to SAFs for all network architectures.

V. Fault Tolerance Techniques

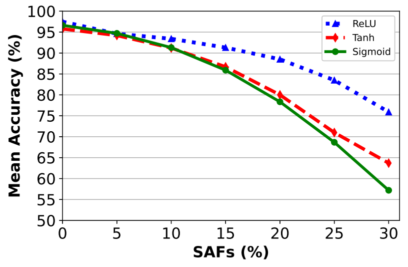
The impact analysis of RRAM defects of Sec. IV highlights the importance of a fault tolerance strategy to mitigate the effects of non-idealities existing in RRAM devices and help in harnessing the full potential of RRAM-based AI accelerators. The proposed fault tolerance strategy is presented in Fig. 5 and consists of two parts, namely, a fault tolerant activation function and a retraining method. The techniques are orthogonal to each other, meaning that they can be applied independently and lead to higher improvements if combined.

A. *Fault tolerant activation function*

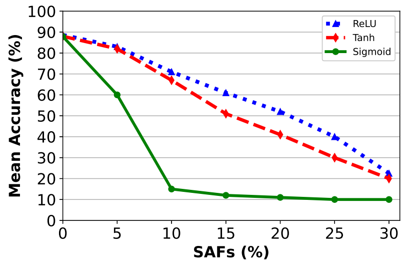
The first part of the fault tolerance flow shown in Fig. 5a, is to choose the proper activation function. To do so, different network instances are created for the selected BNN architecture, one with each activation function, namely, $sigmoid$, $ReLU$, and $tanh$. Then, each BNN instance is trained and mapped on a faulty RRAM crossbar array to assess its fault tolerance capabilities for different fault injection experiments. Finally, the best-performing network instance is selected.

As shown in Figs. 6a and 6b, the BNNs using the $ReLU$ activation function achieve the least classification accuracy drop when SAFs are injected in the crossbar array. Notably, for 10-25$\%$ of SAFs, $ReLU$ scores a 5-10$\%$ improvement in the classification accuracy over $tanh$ and even further over $sigmoid$. This is mainly the case because $ReLU$ activates a neuron only for positive values, thus, resulting in a sparser neuron activity across the network, whereas $sigmoid$ and $tanh$ neurons are not threshold based, and can be active for wide range of weighted input values. This sparser neuron activity helps neurons with ReLU activation function in reducing the propagation of faults to the subsequent layers, as it limits the activation of faulty neurons.

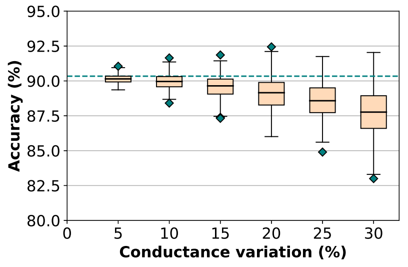
Fig. 6c shows the fault tolerance capability of the 2-layer MNIST BNN with $ReLU$ in the presence of 15% SAFs and conductance variation. Similarly to the results of Sec. IV-B, the impact of conductance variation is rather minimal. For small conductance variation of 5-15$\%$ the additional drop in the classification accuracy is negligible, while we notice a drop of approximately no more than 3% even for high conductance



(a) 2-layer MNIST BNN



(b) Fashion-MNIST BNN



(c) ReLU 2-layer MNIST BNN, 15% SAFs

Fig. 6: SAF and conductance variation impact for different activation functions.

TABLE II: SAF-aware training results.

	2-layer BNN (MNIST)		**4-layer BNN (Fashion-MNIST)**		**VGG (CIFAR-10)**	
SAF	**Classification Accuracy (%)**					
(%)	**Baseline**	**Retrained**	**Baseline**	**Retrained**	**Baseline**	**Retrained**
0	97.3	-	88.2	-	90.1	-
5	96.0	97.3	84.0	87.1	65.0	89.9
10	95.4	97.0	71.0	88.0	10.0	89.7
15	93.0	97.2	62.0	87.9	10.0	89.6
20	89.0	97.1	53.0	88.0	9.0	89.0
25	84.0	97.2	40.0	87.8	8.0	88.7
30	71.0	96.8	22.0	88.5	8.0	89.0

variation. Therefore, $ReLU$ can effectively tolerate this defect. Overall, using $ReLU$ instead of $tanh$ or $sigmoid$ can lead to improvements in the classification accuracy of around 10%.

B. SAF-aware retraining

Performing a fault-aware retraining of a network can enhance its fault tolerance capabilities by learning around the faulty components. To further boost the classification accuracy, our strategy combines the proposed retraining method with the fault tolerant activation function technique described previously.

First, the retraining method identifies the fault distribution across the crossbar array, so that the faulty locations are excluded from the training algorithm and focus on retraining only the weights mapped on fault-free RRAM devices. To achieve this, the gradient mask of Eq. 3 is applied in order to prevent updating the weights corresponding to faulty cells:

$$M_{ij} = \begin{cases} 0, & \text{if } c_{ij} \text{ has SAF} \\ 1, & \text{otherwise} \end{cases} \quad (3)$$

where M_{ij} is the mask element for cell c_{ij} at the junction of the wordline i and the bitline j. Using the gradient mask allows the retraining of the network to converge faster, as the number of weights to be updated is reduced.

Table II presents the classification accuracy improvement of the proposed retraining method using the $ReLU$ activation function for the different BNN architectures. As shown in Table II, SAF-aware retraining is able to almost fully recover the classification accuracy back to the (fault-free) baseline, i.e., 0% SAFs, for all networks and SAF percentages. More precisely, the *recovered accuracy*, i.e., the ratio between the restored and baseline accuracy, can reach up to 99.8%.

VI. Conclusions

Memristor-based CIM is a promising solution to overcome the limitations of conventional computing and deliver energy-efficient architectures for emerging data-intensive application segments. However, memristive devices, such as RRAM, suffer from variability and manufacturing defects. This work investigates the impact of RRAM non-idealities on the recognition capabilities of BNNs and proposes a two-step fault tolerance strategy to alleviate their impact. Results show that the proposed techniques can recover the classification accuracy by 99.8%.

Acknowledgment

This work is funded by EU's Horizon Europe research and innovation programme under grant agreement No. 101070374.

References

[1] H. A. D. Nguyen et al., "A classification of memory-centric computing," *JETC*, 2020.
[2] A. Singh et al., "Low-power memristor-based computing for edge-AI applications," in *ISCAS*, 2021.
[3] A. Gebregiorgis et al., "Dealing with non-idealities in memristor based computation-in-memory designs," in *VLSI-SoC*, 2022.
[4] L. Chen et al., "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *DATE*, 2017.
[5] W. Huangfu et al., "Computation-oriented fault-tolerance schemes for RRAM computing systems," in *ASP-DAC*, 2017.
[6] C. Liu et al., "Rescuing memristor-based neuromorphic design with high defects," in *DAC*, 2017.
[7] L. Xia et al., "Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems," in *DAC*, 2017.
[8] L. Xia et al., "Stuck-at fault tolerance in RRAM computing systems," *JETCAS*, 2018.
[9] A. Chaudhuri et al., "Hardware fault tolerance for binary RRAM crossbars," in *ITC*, 2019.
[10] R. Bishnoi et al., "Special session – emerging memristor based memory and CIM architecture: Test, repair and yield analysis," in *VTS*, 2020.
[11] M. Zahedi et al., "MNEMOSENE: Tile architecture and simulator for memristor-based computation-in-memory," *JETC*, 2022.
[12] A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," *Solid-State Electronics*, 2016.
[13] J. Borghetti et al., "'memristive' switches enable 'stateful' logic operations via material implication," *Nature*, 2010.
[14] S. Yu et al., "RRAM for compute-in-memory: From inference to training," *TCAS-I*, 2021.
[15] A. Hardtdegen et al., "Improved switching stability and the effect of an internal series resistor in HfO 2/TiO x bilayer ReRAM cells," *T-ED*, 2018.
[16] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *Computer Architecture*, 2016.
[17] E. I. Vatajelu et al., "Challenges and solutions in emerging memory testing," *TETC*, 2017.