RRAMedy: Protecting ReRAM-based Neural Network from Permanent and Soft Faults During Its Lifetime

Wen Li[1,2], Ying Wang[1,2], Huawei Li[1,2,3], and Xiaowei Li[1,2]

[1]SKLCA, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]Peng Cheng Laboratory, Shenzhen, China
{*liwen, wangying2009, lihuawei, lxw*}*@ict.ac.cn*

***Abstract*—The emerging memristor technology is considered a promising solution to the edge-oriented deep learning and neuromorphic processor chips because it enables power-efficient Computing-in-Memory (CiM) and normally-off architecture simultaneously. However, as the analog nature and the immature nano-scale fabrication technology, the memristive cells suffer from manufacturing defects, process variations and aging-induced variations, which may incur system and function failures in applications. How to detect and rescue from the permanent and soft faults poses a significant challenge to the edge ReRAM-based deep learning or neuromorphic chips. In this work, we propose an edge-cloud collaborative framework, RRAMedy, to achieve in-situ fault detection and network remedy for memristor-based neural accelerators. In this framework, we present Adversarial Example Testing, a lifetime on-device fault detection technique, which can accurately detect defected cells and memristor soft faults with high probability and at a low cost. Furthermore, the model accuracy can be restored by the proposed edge-cloud collaborative fault-masking retraining and model updating mechanism with a minimized edge-cloud communication overhead. The experimental results show that RRAMedy can effectively detect the memristor permanent and soft faults, protecting the neural accelerator from accuracy and performance degradation in its life cycle.**

I. Introduction

Deep learning technology has proved to be a powerful solution in many critical applications. However, both neural network training and inference demand massive amount of computational resource when deployed on real systems. As the rise of deep learning applications on real-time and power-constrained scenarios, it is natural to leverage the computational power on the edge devices for real-time local DNN inference, while leaving the more computationally-intensive training phase to the cloud services.

Over the past years, many specialized neural accelerator architectures and devices have been studied [1, 2]. Particularly, Resistive Random Access Memory (ReRAM) has become a promising candidate memory technology that embraces the efficiency benefits of near-zero standby power, and non-volatility [3]. Besides, ReRAM crossbar array can not only be used to store data, but also provide in-situ dot product computations as a fascinating Computing-in-Memory (CiM) device. There are plenty of ReRAM-based neural accelerators proposed for energy-efficient applications, like PRIME [1], ISAAC [2].

However, due to the immature nano-scale fabrication technology and the intrinsic nature of memristors, ReRAM cells suffer from both severe permanent faults and soft faults, which will permanently or temporarily change the ReRAM cell state, leading to system performance degradation and even erroneous behavior [4]. Specifically, permanent faults arise from limited endurance of ReRAM technology, which make memristors permanently stuck at high/low resistance and cannot be programmed to represent neural parameters accurately. In contrast, soft faults are transient faults caused by imperfect operations, state-drifts and parameter deviations, due to imperfect manufacture or wear-out mechanism [5]. Besides, soft fault-corrupted cells can be refreshed back to normal values, but such soft faults are subtle to detect and cause unpredictable impacts on neural computation with memristors. It has been demonstrated that soft faults can degrade more than 48% performance of a ReRAM-based neuromorphic system [4]. Hence, due to the existence of both permanent and soft faults, the unreliability of ReRAM memory becomes a concern for ReRAM-based neural accelerator designs.

Several categories of methods have been proposed in prior works to mitigate the effects of manufacture defects and soft faults on ReRAM-based neural computing devices. The first category is on-device training and remedy, which generates a dedicated neural model that can tolerate the faults distributed on the memristor chip [6]. However, practicing on-device training is unrealistic for edge devices lack of computational capability for iterative back-propagation. Besides, even though the model can be pre-trained for once on the specific defected chip before the devices are delivered, it cannot ensure the chip survive the unpredictable in-situ wear-outs and soft faults during its later life cycle. The second category is re-training to

This work was supported in part by the National Natural Science Foundation of China under Grant 61432017, 61876173, 61874124, 61532017, 61504153 and in part by YESS Hip Program by CAST. The corresponding authors are Ying Wang and Huawei Li.

978-1-5386-6648-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCD46524.2019.00020

traditional hardware or software redundancy. Such methods enhance the fault tolerance of deep learning systems by replicating the hardware components or the critical neurons/nodes on the pre-trained models [7]. It incurs performance, memory and energy overhead, and is too expensive to be deployed on the real-time edge systems. The third category is to conduct on-line test and repair routinely in case of faults [8]. However, when and how to perform the "write-verify" scheme is rarely investigated. As the "write-verify" scheme brings additional write operations which will wear-out memristors, it is better to be invoked only when the memory faults appear.

In response to these drawbacks, we propose RRAMedy, a lightweight fault detection and performance recovery framework for the ReRAM-based edge deep learning accelerators. RRAMedy can be performed routinely in spare time during the accelerator's lifetime, targeting on both permanent and soft faults in memristors for the first time. For fault detection, we leverage the fault-sensitivity of Adversarial Examples at first to detect faults with high probability, especially for the drift or read disturbance induced state changes that are hard to detect. Second, we propose a fault-masking model retraining scheme which is aware of the fault-distribution on ReRAM devices and carried on the cloud to recover the system performance to the maximum extent. To summarize, our main contributions are as follows:

- To secure ReRAM-based edge accelerators from fault-induced severe degradation, we investigate the fault models of ReRAM and propose an effective fault detection and recovery framework to protect edge neural accelerators from both permanent faults and soft faults. The framework includes mechanisms of fault detection, cooperative edge-cloud model retraining and model updating, to ensure the functionality of ReRAM-based neural accelerator during its lifetime.
- To efficiently detect either permanent or transient faults, we develop a low-cost software fault detection method for ReRAM-based edge neural accelerators, by leveraging the Adversarial Example training to detect the subtle fault-induced variations, which is hard or expensive for the conventional methods to discover. With the adversarial test set, which is very sensitive to subtle ReRAM cell faults or state drifts during its service cycle, the fault detection probability increases significantly, which prevents the subtle faults from corrupting the results without being noticed.
- For system performance recovery, different approaches are applied to different types of faults. For soft faults, we propose to tune the corrupted cells with local model replication. For permanent defects, an edge-cloud collaborative fault-masking model retraining and updating method is developed to cure the edge devices at a minor overhead.
- We evaluated RRAMedy on a popular ReRAM-based accelerator and modified a state-of-the-art deep learning framework to support fault injection and ReRAM-based performance modeling. It is shown in our experiments that RRAMedy achieves more than 93% detection accuracy and recovers system performance with less than 2% degradation.

We demonstrate RRAMedy with a popular ReRAM-based edge neural accelerator, but our proposed framework can also be applied to other nonvolatile memory and CiM devices.

The rest of the paper is organized as follows. In section II, we introduce the ReRAM-based DNN computing mechenisms and present an overview of the related work. Section III describes the proposed RRAMedy framework in detail. Simulation results are provided in Section IV. Finally, Section V concludes the paper.

II. Background and Motivational Study

A. ReRAM-based DNN Computing

Deep neural network (DNN) is a machine learning architecture which is composed of a series of computational layers and can be represented as a parametric function F:

$$F(x)=f_L(w_L, f_{L-1}(w_{L-1}\dots(f_1(w_1,x)))) \quad (1)$$

wherein x represents the input and f_i refers to the functional layers, including convolutional layers (CONV) and full connected layers (FC).

Nowadays, many specialized deep learning accelerators have been proposed to use ReRAM for edge neural network implementation at the *inference process* [1, 9]. ReRAM cells can not only work as on-chip memory, but also perform in-memory matrix-vector multiplication efficiently. As shown in Fig. 1 (a), memristors are connected as a crossbar structure. When conducting a matrix multiplication $V \cdot G$, the matrix G is programmed as a set of conductance values of the ReRAM cells, while the input vector is represented as a sequence of the analog word-line (WL) voltages v_i. When the voltages are applied to all the WLs simultaneously, the outputs of $V \cdot G$ are sensed as a current set I automatically, achieving in-memory matrix multiplication effectively [10].

Ideally, the weight values are fixed after the training process. However, the unavoidable faults in the memristors will result in weight value fluctuation and further degrade the performance of DNN systems. Taking a CONV layer for

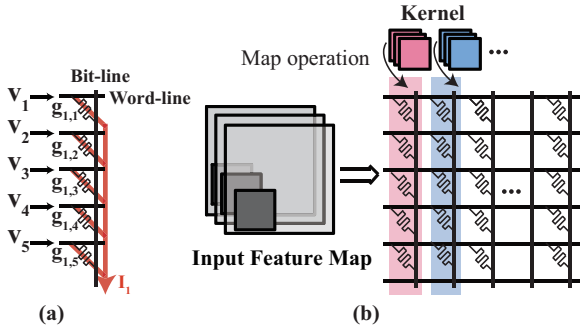

Fig. 1. (a) The analog dot-product computing mechanism of ReRAM. (b) A simple mapping scheme of input feature maps and kernels.

an example, in each convolutional step, a multi-dimensional kernel slides over the input feature maps (IFMs) to extract characteristics and produce the corresponding output feature maps (OFMs). Fig. 1 (b) illustrates a simple weight mapping scheme for convolutional layers. The parameters in the same kernel are reflected as the conductance of the ReRAM cells on a single bit-line. The IFMs are dot-multiplied by the kernel windows and represented as a series of input voltages, preparing to generate OFMs. It is worth pointing out that a specific convolutional kernel w is fixed on the corresponding ReRAM cells and used to compute all output neurons in the corresponding OFM. As such, once a ReRAM cell becomes faulty, it will influence all the values of the corresponding OFM and further propagate layer by layer, resulting in an erroneous output or even system failure [11].

SLC and MLC ReRAM. Recently, many researches work on multi-level cell (MLC)-based deep learning accelerators rather than the traditional single-level cell (SLC) devices. Unlike SLC memories, the MLC memories store a multi-bit value in a single ReRAM cell. However, there are trade-offs in MLC ReRAM memories. In an *n*-level ReRAM MLC cell, the resistance state has to be encoded into *n* levels. The stored values can be changed, even if the states of ReRAM cells have slightly drifted, which significantly increases the unreliability of ReRAM computation. Hence, many deep neural accelerators propose to represent one weight by multiple cells connected to the same WL. Each cell stores one or two bits rather than storing a whole weight in a single cell [12].

B. Related Work

To address the reliability challenges in memristors, many hardware solutions have been proposed to tolerate permanent faults and soft faults. Error Correcting Code (ECC) has been studied in [13] to alleviate the impact of process variations of memristors. However, this technique incurs high penalty with additional power and performance overhead. Besides, a squeeze-search method has been proposed in [8] to identify the ReRAM defects with a March algorithm directly. Though this method is effective, its huge timing overhead prevents it from usage in on-line protection scheme for edge devices. Besides, it brings in additional write operations, which further leads to memory wear-outs.

Considering the huge costs of hardware-based methods, many software-based solutions have been proposed recently. In [14], an offline training method has been proposed with a model mapping strategy. The authors used the prior knowledge of fault distributions to map the weight matrix to memristors and then conducted an offline model retraining to make memristors more resilient to faults by treating the faults as noises during training. However, this method takes no consideration on the fault detection overhead. Moreover, the noise-tolerant models will still face accuracy degradation for some stuck-at faults and severe soft faults, because the training method can improve the robustness of the network model against faults but not completely eliminate the impacts of faults.

To overcome the above-mentioned drawbacks, we propose a general framework which targets both permanent and soft faults and guards ReRAM memory during its service lifetime.

III. The Framework of RRAMedy

In this section, we analyze the fault models of ReRAM-based edge neural accelerators and present an edge-cloud cooperative model rescuing scheme, RRAMedy, to prevent the classification accuracy decrease from the unreliable memristors for ReRAM-based deep learning systems.

A. Target Fault Models

ReRAM's distinctive characteristics come with reliability concerns. The working mechanism of ReRAM relies on the generation and rupture of the oxygen ions (O^{2-}) and oxygen vacancies (V_O). The stochastic nature of V_O makes ReRAM susceptible to many reliability problems [4], including:

Permanent Faults. Permanent faults (Hard Faults) of ReRAM cells force the resistance states fixed at high resistance (stuck-at-0 fault) or low resistance (stuck-at-1 fault), which are usually caused by fabrication defects [8] and limited endurance.

Soft faults. The ReRAM soft faults are mainly resulted from: a) the unavoidable degradation mechanisms and wear-out mechanism, b) manufacturing defects, especially the imperfect "electroforming" process [10, 15]. These soft faults can be observed as retention failures, read disturbance or write disturbance [5, 16]. There are two classical ReRAM structures, including a 1T1R (one transistor one memristor) structure and a crossbar structure. For the crossbar structure, ReRAM cells are connected directly without transistors. When a read/write voltage is applied on a word-line/bit-line, the unselected memristors on the same word-line/bit-line may be affected, which can lead to read/write disturbance. Although the 1T1R ReRAM structure typically uses an access transistor to avoid the sneak-path problem and for memristor programming, it will still have the risks of retention failures. It is resulted from the behavior of V_O and O^{2-} in conductive filaments and can be accelerated with high temperature. Even though the ReRAM-based chips have passed the manufacturing test, cells will still suffer from faults/variations during its lifetime, and these effects can be accumulated to result in a data disturbance [17].

In a nutshell, ReRAM-based edge neural accelerators face both inevitable permanent faults and soft faults in practice. Permanent faults appear over memory lifetime and cannot be tuned. Soft faults have no permanent corruption on the stored data, but they still have the ability to damage the system. Hence, it is necessary to detect the faults and rescue the system performance from them.

B. Overview

We present the overview of RRAMedy firstly. As illustrated in Fig. 2, we model the edge deep learning scenario with two parties, a *cloud server* and an *edge device* with ReRAM-based memory. The edge device routinely detects its fault

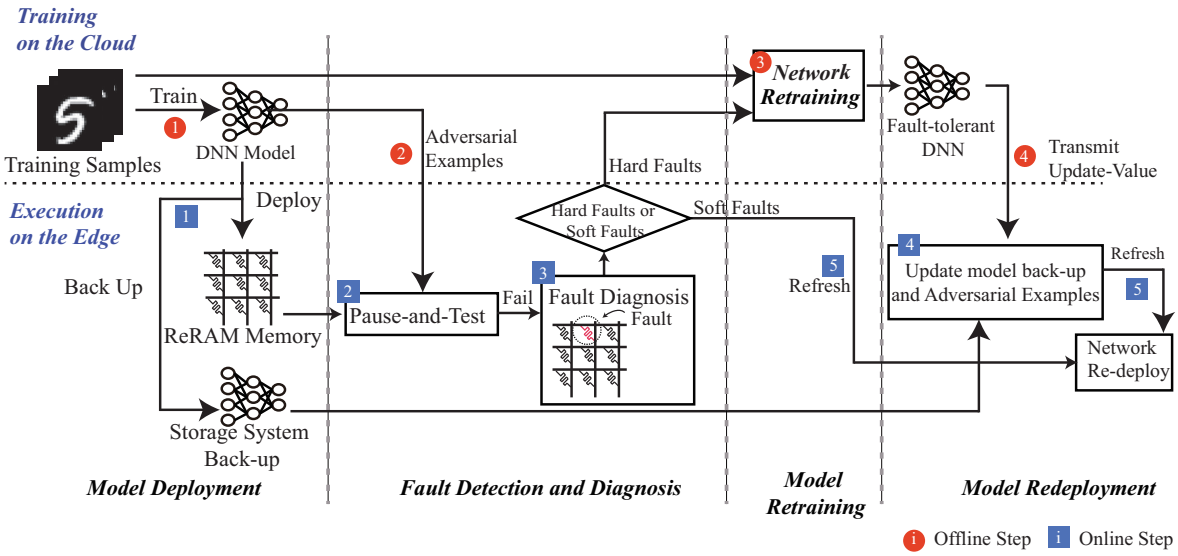

Fig. 2. The global flow of the RRAMedy framework.

occurrence with the proposed Adversarial Example Testing (AET) method. Once a device detects unrepairable faults and requires updating the current model parameters, the cloud server will retrain a device-specific model that tolerates the faults in the ReRAM memory via a fault-aware retraining step and send the model updates back to the device. Finally, the device incorporates these updates into its original model parameters to maintain the system performance through the model resilience.

The RRAMedy framework consists of two primary components, including (1) a fault *detector* and (2) an in-cloud network *retrainer*. The fault *detector* consists of a "Pause-and-Test" (P&T) mechanism and a "Fault Diagnosis" (FD) component. The P&T mechanism is periodically invoked to detect system accuracy degradation caused by ReRAM state variations, while the FD component is used for fault diagnosis and generates the corresponding fault distribution. After receiving the hard fault distribution from the edge, the in-cloud network *retrainer* finetunes the network model to tolerate the irreparable defects adaptively. Leveraging the excellent self-recovering capability of neural networks, the faulty network can be retrained with the existence of unrepairable hard defects. Fig. 2 demonstrates the general flow of RRAMedy framework that includes four primary phases with four online steps and five offline steps.

Model Deployment. Firstly, the cloud server trains a network on the cloud (step ❶) and transmits it to the edge device. Then, the edge device deploys the model on the deep learning accelerator for execution and also makes a backup on the storage system (step 1).

Fault Detection and Diagnosis. As having addressed, fault detection and diagnosis always bring high overhead. To reduce the overhead, a routinely invoked fault detection mechanism is established on the edge device. As seen in Fig. 2, the server generates and selects a set of adversarial examples for fault detection. These generated adversarial examples are transmitted, stored in the storage system of devices (step ❷) and periodically fed into the ReRAM accelerator for the fault detection routine (step 2). The detection result will be further analyzed to instruct the execution of the FD component for accurate fault location (step 3). Specifically, if the ReRAM accelerator fails to generate correct predictions on the adversarial test set, the P&T component will raise an alarm flag to trigger FD. If there is no permanent fault located, it means the ReRAM accelerator encounters soft faults. The device will refresh the corrupted cells with model back-up (step 5). Otherwise, a permanent fault is found by the FD component, and the fault map is sent to the cloud for model retraining.

Model Retraining. The server waits for the edge devices to report its fault situation. As the fault maps are received, the server will retrain the neural network with the proposed fault-masking method and adaptively adjust the neural network to tolerate the device-specific faults (step ❸).

Model Redeployment. To reduce the communication overhead, the server only transmits the quantized weight update values to the edge device (step ❹). The edge device will then update the cloud-retrained networks on its storage system (step 4) and use the updated backup to refresh the ReRAM states to mitigate the fault-induced accuracy degradation. The backup is also used to refresh the soft fault-induced struck cells once detected (step 5).

C. Adversarial Example Testing on the Edge

It is well-known that the fault detection and location process is time-consuming, which makes it unacceptable to perform periodically, especially for the edge ReRAM expensive to program due to write overhead. Based on this, we pursue to find a more realistic method to capture the fault-induced system behavior failures. Only when the deep learning system is detected with behavior deviations, the FD function will be triggered, which significantly minimizes the system overhead.

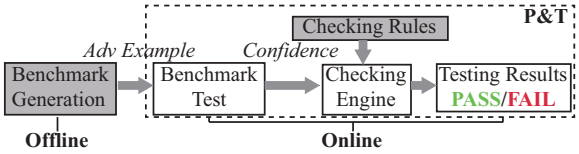

Fig. 3. The overview of Pause-and-Test mechanism (grey components can be adjusted by the cloud servers).

The **Pause-and-Test mechanism** (P&T) is proposed to periodically analyze the fault existence at the system behavior level. Fig. 3 illustrates the high-level view of the P&T mechanism. It can be described as a function: $M \rightarrow \{0, 1\}$, that decides whether the neural model M is heavily affected by the faults or resistance variations in the ReRAM memory. Unlike traditional systems, which require bit-level comparison to detect system faults, the deep learning systems should analyze the output confidence score directly, because the bit-by-bit comparison is unnecessary. Hence, we consider a simple strategy to distinguish the faulty models from normal models: we feed the test benchmark into the edge neural systems and compare the confidence score of the actual model's prediction with the original prediction confidence. The deviations of the original and actual prediction confidence score should be negligible. Once the prediction scores are determined to be heavily different based on the predefined checking rules, the model parameters are likely to be corrupted and the ReRAM memory is most likely suffering from faults or defects. Then, the RRAMedy framework will further diagnose the faults or variations with the FD process.

However, neural networks are thought possessing the intrinsic resilience to errors and noises of certain distribution in either inputs and neural weights. Randomly picked input test samples may not activate the faults in ReRAM cells at all, and make the faults escape from software testing, which will increase the risk of fatal failure caused by the latent faults in critical tasks. Thus, we have to propose a more sensitive test method that will activate and detect the faults and elusive cell state variations with high coverage and probability.

Recently, an adversarial example generating method has been proposed in the deep learning security domain, which is called FGSM (Fast Gradient Sign Method) [18]. It can be described as:

$$x' = x + \varepsilon \cdot sign(\nabla x J(F(x), y)) \tag{2}$$

where the adversarial example x' is generated by adding perturbations to the original sample x, as shown in Fig. 4. The perturbations $sign(\nabla x J(F(x), y)$ are calculated as the sign of the gradient of the model's loss function $J(\cdot, \cdot)$, pushing the original input move towards the direction of the gradient.

Though the adversarial examples are used to mislead DNNs originally, we find that they also have great performance on fault detection. Essentially, the adversarial examples are elaborately generated according to the loss gradient, which is on the contrary of the weights update direction. Thus, the weight variations will have a severer impact on the prediction of adversarial examples.

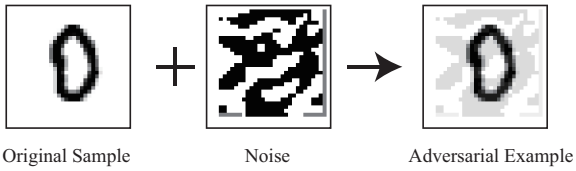

Fig. 4. A demonstration of adversarial example generation.

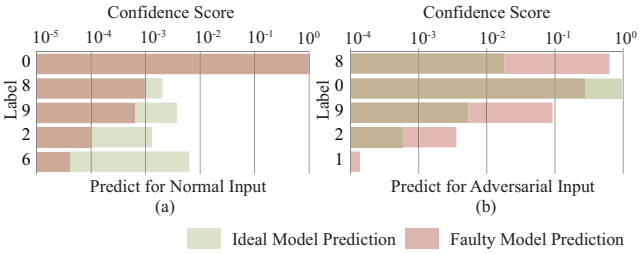

Fig. 5. (a) The variations of confidence score when feeding the normal input into the normal network and faulty network. (a) The variations of confidence score when feeding the adversarial example into the normal network and faulty network.

To demonstrate the sensitivity of adversarial examples on parameter variations, we conducted an experiment on the MNIST dataset with LeNet network. We created a faulty model by injecting noises on the original model parameters, degrading the accuracy from 98.6% to 96.8%. Fig. 5 shows the fault-caused confidence deviations of the original model and faulty model with the same input. We observe that the deviation of top-ranked confidence score is 4% by using the original input, while the prediction totally changes from label '0' to label '8' by using the adversarial example. This is because that the adversarial examples are generated elaborately according to the loss gradient. Small disturbance on network parameters will result in large deviations on confidence score. Hence, it is easier to detect the subtle faults on edge devices by using adversarial examples.

Fault Diagnosis. Here, we adopt the March C$^-$ test algorithm for further fault diagnosis. Specifically, March C$^-$ applies a series of read/write operations to a given memristor array by a specific address order and can achieve complete fault coverage by analyzing the fault dictionary [19]. March C$^-$ is denoted as follows:

$$\begin{aligned}\text{March C}^- - \{&\Updownarrow(\text{w0}); \Uparrow(\text{r0}, \text{w1}); \Downarrow(\text{r1}, \text{w0});\\ &\Downarrow(\text{r0}, \text{w1}) \Downarrow(\text{r1}, \text{w0}); \Uparrow(r0);\}\end{aligned} \tag{3}$$

The symbol '$\Uparrow$' , '$\Downarrow$' and '$\Updownarrow$' denote the order of address sequence. The increasing address direction is represented by the '$\Uparrow$' symbol, and the decreasing address direction is denoted by the '$\Downarrow$' symbol. The symbol '$\Updownarrow$' is used when the address direction is irrelevant. Besides, '$w0$', '$w1$', '$r0$' and '$r1$' represent the write 0, write 1, read 0 and read 1 operation, respectively. It has been proved that the six March elements in the March C$^-$ algorithm can detect all the modeled faults [20]. Obviously, it requires five read operations and five write operations for each memristor which brings huge time overhead. Considering the limited write endurance of ReRAM, our proposed RRAMedy framework only activates the FD component when the P&T mechanism detects the memristor

faults, instead of using the March algorithm directly for fault detection.

D. Fault-masking Retraining on the Cloud

Once there are unrepairable faults detected on the edge devices, the cloud servers need to take measures to rescue the system performance from the memory faults. Conventional edge-based model retraining solutions have an obvious weakness: the retraining process consumes high hardware resources, making it unpractical to be deployed on the edge device. There are also off-device methods that are carried through model training, but it only focuses on making the network robust to faults, rather than eliminating the impacts of faults [14]. Based on this observation, we explore a cloud-edge collaborative model retraining method.

For the **Cloud-Edge Collaborative Method**, the edge device only needs to generate the corresponding fault distribution as a fault-mask in "Fault Detection and Diagnosis" phase (Fig. 2, step 3) and transmit it to the cloud server. Then, the cloud server will leverage the received fault-mask and apply the proposed **fault-masking retraining method** to adaptively adjust the neural network to tolerate the device-specific faults.

Unlike previous work that enhances the robustness of network model by using specialized regularization in training, in this work, the goal of the offline model retraining process is to construct a fault-tolerate network $F'(x)$, which can recover the classification accuracy from faulty edge devices. The output of $F'(x)$ is supposed to be close to the original neural network output $F(x)$, that is:

$$\forall x : \min \Delta(F(x), F'(x)) \tag{4}$$

It has been proven that DNN has the inherent self-recovery capability to relearn the ground truth from the corrupted weights [21]. By applying the Back Propagation (BP) algorithm to update weights, the training model parameters can self-adapt the faults iteration by iteration. The weight updating process can be derived as:

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i} \tag{5}$$

wherein, η refers to the learning rate, E denotes the global error. However, the weight w_i is tuned to achieve high accuracy without consideration to be adjusted to adapt permanent faults. To reduce the performance degradation caused by the occurrence of unrepairable permanent faults, we propose to mask the faulty weights which suffer from "stuck-at" faults during the model retraining phase. Specifically, the update of weight w_i can be described as:

$$w_i \leftarrow Mask(w_i - \eta \frac{\partial E}{\partial w_i}) \tag{6}$$

The $Mask$ function is used to fix the faulty bits to their stuck-at values during the training phase, according to the received fault-mask. For example, as shown in Fig. 6, there is a 16-bit-width weight mapped on a row of memristors. However, a stuck-at-1 fault and a stuck-at-0 fault occur on these memristors simultaneously. To ensure the retraining phase can tolerate specific weight deviations, we mask the

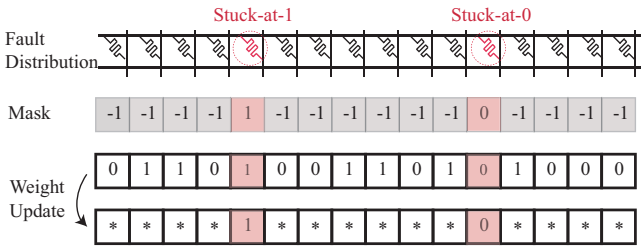

Fig. 6. Fault-masking retraining.

erroneous bits based on the bit-memristor mapping. When the bit value is mapped on a cell with stuck-at-1 (0) fault, we will fix the value to 1 (0) during the training phase. Therefore, by using the $Mask$-based retraining method on the cloud server, the network itself can compensate for the performance degradation from the error induced by the ReRAM faults.

Due to communication resource and time constraints, the cloud servers only transmit the gradients to the edge devices, which are also quantized to further reduce the communication overhead, as it is shown that the gradients can be precisely represented by sparse and lower-bit code [22]. Then the edge device will use these gradients to update its local model back-ups and refresh the ReRAM states to reduce the fault-induced accuracy degradation. To rescue from soft faults, the edge device will employ the back-up neural model to refresh the ReRAM arrays. Here, we propose to employ iterative write on the device to make sure the cells are correctly programmed even when the cells are having parametric fluctuations [23].

IV. Evaluations

A. Experimental Set-up

Datasets and Workload. We investigate the effectiveness of RRAMedy on two standard datasets, MNIST and Cifar-10, with three different network architectures, as described in Table I. The MNIST dataset is used for hand-written digit recognition with 70,000 gray-scale images, wherein 60,000 images are used for training and 10,000 images are testing data. The Cifar-10 dataset consists of 60,000 true-color images of size 3×32×32. The dataset is divided with 50,000 training images and 10,000 test ones.

Fault Injection mechanism. To precisely model the impacts of unreliable ReRAM cells on the accuracy of neural networks, we modified the Caffe framework for fault injection simulation to inject the real-world ReRAM-based faults into ReRAM cells and propagate the errors from the device level to the applications. The faults are injected randomly in the proper network parameters by modifying the 16-bit fixed-point weights in the simulator.

TABLE I
Benchmarks

Network	Dataset	Classes	Architecture	Accuracy
MLP	MNIST	10	3 FC	0.9616
LeNet	MNIST	10	2 CONV + 2 FC	0.9858
ConvNet-quick	Cifar-10	10	3 CONV + 2FC	0.745

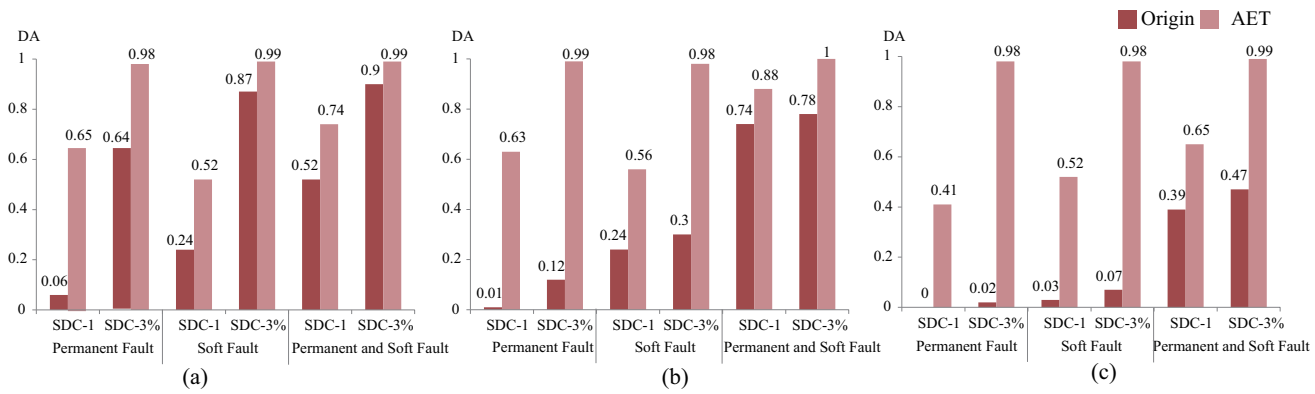

Fig. 7. Detection accuracy on SLC ReRAM on (a) MLP (b) LeNet (c) ConvNet network with the consideration of only permanent faults; only soft faults and both permanent faults and soft faults occurence.

B. *Effectiveness of Adversarial Example Testing*

We propose to detect the fault occurrence in deep learning systems with two detection criteria from [11], including:

SDC-1: When the top-ranked prediction of the executed DNN is different from the fault-free prediction, we consider that there exist faults in the edge neural accelerator.

SDC-3%: The top-ranked confidence score is compared with the ideal execution. If the variations are more than +/- 3%, we consider that the ReRAM accelerator is faulty.

To further evaluate the effectiveness of our Adversarial Example-based detection, we did experiments on all the three above-mentioned networks. We tested 100 faulty-models for each network and simulated on both SLC and MLC ReRAM.

Here, we define the **Detection Accuracy (DA)** as a measure of how well the fault-affected neural network can be differentiated from the original model. Specifically, it is defined as the accuracy of the detector when identifying the faulty networks, and can be formulated as:

$$\mathrm{DA} = \frac{\text{The Number of Identified Faulty Models}}{\text{Total Number of Tested Faulty Models}} \tag{7}$$

Furthermore, we try to choose the best adversarial example for fault occurrence detection. We tested different disturbance ε (Equation 2) to generate adversarial examples. The ε used for the AET method is shown in Table II.

(a) Evaluations on SLC ReRAM

For the SLC mode of ReRAM-based deep learning accelerator, a 16-bit fixed-point weight needs to be stored in 16 memristors. Considering each cell may suffer from memory faults, we injected both permanent faults and soft faults by randomly modifying bit values within a weight. For permanent faults, five-thousandths of the weight bits are injected with stuck-at faults in LeNet and MLP network. While for the ConvNet network, it is more sensitive to faults. Even if only one-thousandth of weights are faulty, the classification accuracy drops from 74% to 65% sharply. Since we need to detect faults before they become uncontrolled, we only injected 0.2‰ faults on ConvNet weight values. As for soft faults, we injected 1% faults into the MLP and LeNet network, and 0.4‰ faults into the ConvNet network. Furthermore, we also tested the DAs of the proposed detection method when both permanent faults and soft faults occur simultaneously.

TABLE II
THE ε USED IN AET METHOD

Network	MLP	LeNet	ConvNet
ε	0.022	0.095	2.1

Fig. 7 illustrates the DAs of two detection methods on three neural networks. "Origin" is the strategy that uses the normal input for fault detection, while "AET" uses the proposed AET method. For example, the "Origin" method only achieves 64% detection accuracy on the MLP network for permanent faults, 87% for soft faults, and 90% for the existance of both hard and soft faults with the SDC-3% criterion (Fig. 7 (a)). But by using the AET method, the detection accuracy achieves more than 98% in all the three faulty conditions. Furthermore, since SDC-1 is a less strict criterion than SDC-3%, smaller changes can be detected by using SDC-3%. Hence, we focus on SDC-3% in the rest of the paper to pursue higher detection accuracy.

Fig. 7 (b) and Fig. 7 (c) shows the DAs on convolutional neural networks, LeNet and ConvNet. Obviously, the AET method outperforms the "Origin" test method by more than 22%. Besides, it is worth noting that when the ReRAM-based deep learning accelerators suffer from both permanent faults and soft faults simultaneously, the AET method achieves more than 99% detection accuracy.

(b) Evaluations on MLC ReRAM

For the MLC mode of ReRAM-based deep learning accelerator, we consider a 2^2-level MLC as [2] has used in this section. Each 16-bit weight is distributed to eight ReRAM MLCs. For permanent fault occurrence simulation, the injected faulty rate is as the same as the SLC mode. For soft fault occurrence simulation, the resistance variations are injected as follows:

$$w_i = w_i + \theta_i; \theta \sim (0, \sigma^2) \tag{8}$$

Here, the θ are set to 0.01 (small), 0.03 (medium) and 0.05 (high) respectively to simulate resistance variations.

For permanent faults detection, as shown in Fig. 8, the "Origin" method can hardly differentiate the faulty models from original models, but the AET method achieves more than 97% detection accuracy with the elaborately generated

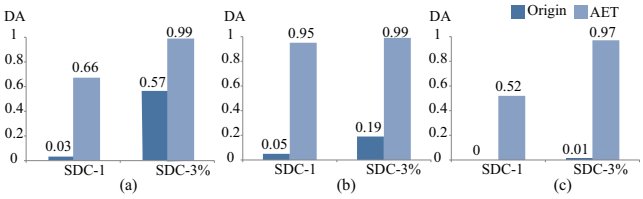

Fig. 8. Permanent fault detection accuracy with the implementation on MLC ReRAM of (a) MLP, (b) LeNet, (c) ConvNet network

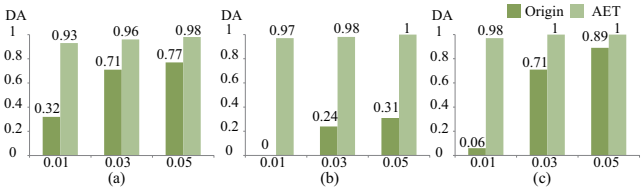

Fig. 9. Soft fault detection accuracy with the implementation on MLC ReRAM of (a) MLP, (b) LeNet, (c) ConvNet network (The X-axis represents the resistance variations θ).

adversarial example. For soft faults detection, as shown in Fig.9, with larger resistance variations, the DA increases for both three networks. This is because that larger resistance variations will cause severer performance reduction and will lead its output confidence change obviously. When considering that both permanent faults and soft faults occur simultaneously, Fig. 10 shows that by using AET method, more than 96% faulty models can be detected on all three networks. Hence, the proposed AET dramatically achieves high detection accuracy on all the three tested fault occurrence situations.

Performance comparison. To compare the performance of the AET method with traditional March C$^-$ algorithm, we execute our benchmarks on a CNN accelerator similar to ISAAC [2], running at 1.2 GHz. The 16-bit fixed point weight is split into eight 2-bit memristors and the crossbar is composed of 128 * 128 ReRAM cells [24].

As shown in Table III, The proposed AET method achieves a speedup from 13× to 1062× in comparison with the traditional March C$^-$ algorithm. Since the full-collected layer has less computational operations but more occupied parameter storage space, for benchmarks with more full-collected layers, the AET method has a higher speed-up ratio. Besides, as the write endurance of ReRAM is limited, the proposed AET method saves the memristors from unnecessary memory wear-outs.

C. Effectiveness of Online Retraining

We evaluated the proposed fault-masking training method by simulation-based fault injection. We injected both stuck-at-0 faults and stuck-at-1 faults on each benchmark. The tested fault rate varies from 0.005 to 0.015 on MLP and LeNet, while 0.001 to 0.003 for ConvNet. Fig. 11 presents the retrieved accuracy for all the three networks with the SLC ReRAM

TABLE III
THE PERFORMANCE SPEEDUP OF AET METHOD

	MLP	LeNet	ConvNet
Speedup	1062×	202×	13×

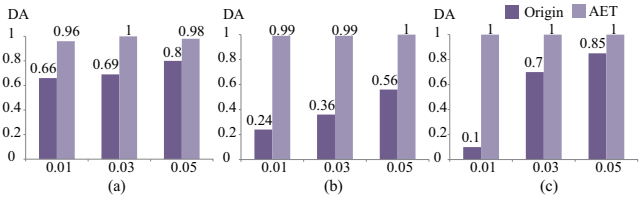

Fig. 10. Fault detection accuracy, considering both permanent faults and soft faults on MLC ReRAM with (a) MLP, (b) LeNet, (c) ConvNet network (The X-axis represents the resistance variations θ).

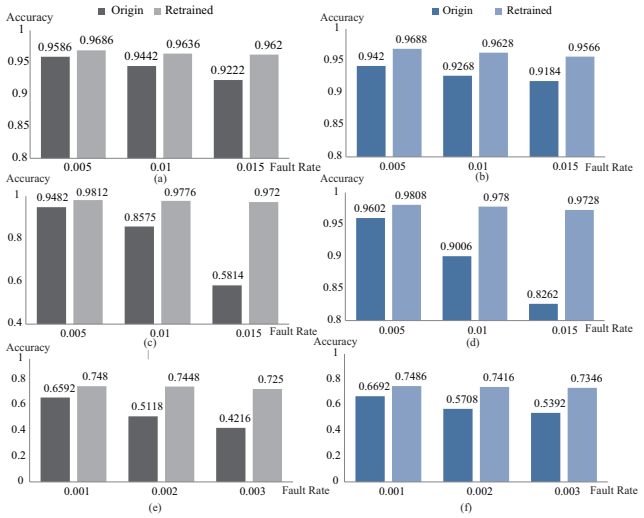

Fig. 11. Retrieved accuracy of fault-masking training with SLC ReRAM implementation ((a) MLP (c) LeNet (e) ConvNet network) and MLC ReRAM implementation ((b) MLP (d) LeNet (f) ConvNet network)

implementation and the MLC ReRAM implementation respectively. As shown in Fig. 11, the ConvNet is significantly affected by parameter variations. Even though it suffers only one-tenth of injected faults of other two benchmarks, the accuracy is still dropped by about 8%. The reason is that the Cifar dataset is more complex than MNIST dataset. Besides, ConvNet only has a small series of layers which makes it has limited fault-tolerance capability. Furthermore, we can conclude from the results that, accuracy can retrieve from heavily system degradation by leveraging the proposed fault-masking retraining method. After retraining, the system performance degradation is less than 2% on all the three benchmarks.

V. CONCLUSION

In this paper, we firstly analyze the unreliability of ReRAM-based edge neural accelerators and present RRAMedy, a novel framework to survive system performance from both permanent faults and soft faults on ReRAM chips. For fault detection, we introduce a lightweight Adversarial Example Testing method to detect the subtle fault-induced variations. For retrieving the system performance, we consider the limited resources on edge devices and propose an edge-cloud collaborative fault-masking retraining method to mitigate the model re-learning phase to the cloud. The experimental results show that the RRAMedy has high detection accuracy and can recover the recognition accuracy with little performance degradation.

REFERENCES

[1] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proc. ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 27–39, June 2016.

[2] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 14–26, June 2016.

[3] H.-S. P. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. T. Chen, and M. Tsai, "Metal-oxide RRAM," *Proceedings of the IEEE*, vol. 100, pp. 1951–1970, June 2012.

[4] A. M. S. Tosson, S. Yu, M. H. Anis, and L. Wei, "Analysis of RRAM reliability soft-errors on the performance of RRAM-based neuromorphic systems," in *Proc. 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 62–67, July 2017.

[5] S. Swami and K. Mohanram, "Reliable nonvolatile memories: Techniques and measures," *IEEE Design & Test*, vol. 34, pp. 31–41, June 2017.

[6] B. Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang, "Vortex: Variation-aware training for memristor x-bar," in *Proc. ACM/EDAC/IEEE 52nd Design Automation Conference (DAC)*, pp. 1–6, June 2015.

[7] L.-C. Chu and B. W. Wah, "Fault tolerant neural networks with hybrid redundancy," in *Proc. 1990 IJCNN International Joint Conference on Neural Networks*, vol. 2, pp. 639–649, June 1990.

[8] C. Chen, H. Shih, C. Wu, C. Lin, P. Chiu, S. Sheu, and F. T. Chen, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Transactions on Computers*, vol. 64, pp. 180–190, Jan 2015.

[9] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *Proc. 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 541–552, Feb 2017.

[10] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Fault tolerance in neuromorphic computing systems," in *Proc. ACM/IEEE 24th Asia and South Pacific Design Automation Conference (ASPDAC)*, pp. 216–223, 2019.

[11] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proc. ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, (New York, USA), pp. 8:1–8:12, 2017.

[12] Y. Long, X. She, and S. Mukhopadhyay, "Design of reliable dnn accelerator with un-reliable reram," in *Proc. 2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1769–1774, March 2019.

[13] D. Niu, Yang Xiao, and Yuan Xie, "Low power memristor-based reram design with error correcting code," in *Proc. 17th Asia and South Pacific Design Automation Conference (ASPDAC)*, pp. 79–84, Jan 2012.

[14] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar," in *Proc. Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 19–24, March 2017.

[15] P. Pouyan, E. Amat, and A. Rubio, "Reliability challenges in design of memristive memories," in *Proc. 5th European Workshop on CMOS Variability (VARI)*, pp. 1–6, Sep. 2014.

[16] S. Hamdioui, P. Pouyan, H. Li, Y. Wang, A. Raychowdhur, and I. Yoon, "Test and reliability of emerging non-volatile memories," in *Proc. IEEE 26th Asian Test Symposium (ATS)*, pp. 170–178, Nov 2017.

[17] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Fault tolerance for RRAM-based matrix operations," in *Proc. 2018 IEEE International Test Conference (ITC)*, pp. 1–10, Oct 2018.

[18] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. International Conference on Learning Representations*, 2015.

[19] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[20] P. Liu, Z. You, J. Kuang, Z. Hu, H. Duan, and W. Wang, "Efficient march test algorithm for 1t1r cross-bar with complete fault coverage," *Electronics Letters*, vol. 52, no. 18, pp. 1520–1522, 2016.

[21] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.

[22] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in Neural Information Processing Systems*, pp. 1509–1519, 2017.

[23] K. Jo, C. Jung, K. Min, and S. Kang, "Self-adaptive write circuit for low-power and variation-tolerant memristors," *IEEE Transactions on Nanotechnology*, vol. 9, pp. 675–678, Nov 2010.

[24] X. Dong, *Modeling and Leveraging Emerging Non-volatile Memories for Future Computer Designs*. PhD thesis, The Pennsylvania State University, 2011.