Training-Free Stuck-At Fault Mitigation for ReRAM-Based Deep Learning Accelerators

Chenghao Quan, Mohammed E. Fouda, *Senior Member, IEEE*, Sugil Lee, Giju Jung, Jongeun Lee, *Member, IEEE*, Ahmed E. Eltawil, *Senior Member, IEEE*, and Fadi Kurdahi, *Fellow, IEEE*

***Abstract*—Although Resistive RAMs can support highly efficient matrix–vector multiplication, which is very useful for machine learning and other applications, the nonideal behavior of hardware, such as stuck-at fault (SAF) and IR drop is an important concern in making ReRAM crossbar array-based deep learning accelerators. Previous work has addressed the nonideality problem through either redundancy in hardware, which requires a permanent increase of hardware cost, or software retraining, which may be even more costly or unacceptable due to its need for a training dataset as well as high computation overhead. In this article, we propose a very lightweight method that can be applied on top of existing hardware or software solutions. Our method, called forward-parameter tuning (FPT), takes advantage of a certain statistical property existing in the activation data of neural network layers, and can mitigate the impact of mild nonidealities in ReRAM crossbar arrays (RCAs) for deep learning applications without using any hardware, a dataset, or gradient-based training. Our experimental results using MNIST, CIFAR-10, and CIFAR-100, and ImageNet datasets in binary and multibit networks demonstrate that our technique is very effective, both alone and together with previous methods, up to 20% fault rate, which is higher than even some of the previous remapping methods. We also evaluate our method in the presence of other nonidealities, such as variability and IR drop. Furthermore, we provide an analysis based on the concept of the effective fault rate (EFR), which not only demonstrates that EFR can be a useful tool to predict the accuracy of faulty RCA-based neural networks but also explains why mitigating the SAF problem is more difficult with multibit neural networks.**

***Index Terms*—Accelerator, artificial neural network, batch normalization (BN), ReRAM crossbar array, stuck-at fault (SAF).**

Manuscript received 6 March 2022; revised 30 June 2022, 15 September 2022, and 19 October 2022; accepted 21 October 2022. Date of publication 15 November 2022; date of current version 20 June 2023. This work was supported in part by IITP Grant through Artificial Intelligence Graduate School Program (UNIST) under Grant 2020-0-01336, IITP Grant through ITRC Support Program under Grant IITP-2021-0-02052, and NRF under Grant 2020R1A2C2015066 funded by MSIT of South Korea; and in part by the Free Innovative Research Fund of UNIST under Grant 1.170067.01. The EDA tool was supported by the IC Design Education Center (IDEC), South Korea. This article was recommended by Associate Editor Y. Li. *(Corresponding author: Jongeun Lee.)*

Chenghao Quan, Sugil Lee, Giju Jung, and Jongeun Lee are with the Department of Electrical Engineering, Ulsan National Institute of Science and Technology, Ulsan 44919, South Korea (e-mail: jlee@unist.ac.kr).

Mohammed E. Fouda and Fadi Kurdahi are with the Center for Embedded and Cyber-Physical Systems, University of California at Irvine, Irvine, CA 92697 USA.

Ahmed E. Eltawil is with the CEMSE Division, King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia.

Digital Object Identifier 10.1109/TCAD.2022.3222288

I. Introduction

As ARTIFICIAL intelligence and neural networks have become more widely used in many diverse applications, there is also a growing interest in hardware architectures that can accelerate them. ReRAM crossbar arrays (RCAs) are a promising technology that can offer extremely fast matrix–vector multiplication (MVM), which is a crucial operation in many deep neural networks (DNNs). Typically, a weight matrix is mapped as conductance to RCA memristors whereas input activations are mapped as voltage to input vector, and the resulting output current represents the MVM result via Ohm's law. Such RCA-based DNN accelerators (e.g., [1], [2], [3], [4], and [5]) comprise of many RCAs, due to the limited size of an RCA and the difficulty of reprogramming RCA memristors for different weight parameters.

Despite its promising effectiveness, RCAs come with some challenges, such as high peripheral circuit overhead (e.g., analog-to-digital converters [6]) and functional inaccuracy due to device and circuit nonidealities. In particular, nonidealities in RCAs, such as stuck-at fault (SAF) [7], [8], IR drop [9], [10], [11], [12], and device variabilities [13], [14], have been shown to degrade the quality of application results severely. In this article, we focus on SAF, which is a very common problem where a memristor is permanently set to either a high-resistance state (HRS) or a low-resistance state (LRS). We refer to the case where a memristor is permanently stuck to LRS as *stuck-at-one (SA1)* and the other as *stuck-at-zero (SA0)*.

Previous work on mitigating the effect of SAF for RCA-based DNN accelerators includes hardware and software approaches. The hardware approach [15], [16] typically adds redundancy in RCA hardware such as adding an extra row [15] which can be used to remap the faulty memristor values or recover the correct output values. However, such a hardware method comes with permanently increased hardware cost along with higher energy consumption. The hardware redundancy methods (e.g., [15] and [16]) add extra rows to recover the correct output values, which requires larger crossbar arrays. While the cost increase due to extra rows may seem inexpensive, having extra rows also requires additional circuitry to correctly route the input for the extra rows. Moreover, such methods depend on the extra rows being free of SAF, which is hard to guarantee in general. In addition, remapping [16] requires suppressing the current contribution from

the now unused rows, which can add to the hardware overhead. Recovery method [15], on the other hand, requires detecting the output current for each RCA. Such a detection step must be done for all of the testing data to get the average difference of the current shift between real output and ideal output, as well as reprogramming the extra row for each RCA to correct output values, all of which could be expensive.

The most straightforward software approach [17], [18], [19] is to retrain the DNN with the knowledge of the SAF information in the hardware. However, such a retraining must be done for each DNN accelerator, leading to a large computation cost, since the SAF pattern must be different from accelerator to accelerator. Also, even detecting the SAF pattern [20], [21], which is necessary for SAF-aware retraining, can be a costly operation because there can be hundreds, if not thousands or more, RCAs in a typical accelerator. Furthermore, acquiring a train dataset may be difficult, and reprogramming after retraining can be another very time-consuming step. Though some researchers have proposed online retraining [21], implementing backpropagation and weight update in hardware is very costly, not to mention the energy and endurance issue with frequent write operations to memristors.

In this article, we propose a novel approach called forward parameter tuning (FPT), which is similar to retraining (as it updates network parameters) but does not require datasets, additional hardware, gradient-based training, or any backward pass.[1] Unlike remapping, which is completely agnostic to the kind of computation being performed, our technique exploits the statistical property of DNN computation, and is hence complementary to remapping techniques. Our technique mainly targets binary ReRAM devices as they are more mature and practical with minimal variability issues but it also works well on multibit ReRAM devices.

Our experimental results demonstrate that our FPT method can increase the resilience of ReRAM-based accelerators without retraining, achieving offline-retraining-level accuracy even at 20% fault rate (FR) in the binarized neural network (BNN) and quantized neural network (QNN). FPT also produces better results in comparison with, and on top of, previous methods [17], [23], [24]. We also evaluate our method in the presence of other nonidealities, such as variability and IR drop. Finally, we present an effective fault rate (EFR) analysis, which not only demonstrates that EFR can be a useful tool to predict the accuracy of faulty RCA-based neural networks but also shows that SAF is more a serious problem in multibit neural networks.

The remainder of this article[2] is organized as follows. After discussing related work in Section II, we present our FPT method in Section III, and provide an analysis of our method based on the concept of EFR in Section IV. Section V presents our experimental results, and this article concludes in Section VI.

[1]It was called *Free Parameter Tuning* in the conference version [22], but we rename it to *Forward Parameter Tuning* to emphasize the fact that it only involves forward computation.

[2]This article extends our earlier conference paper [22] as follows. Section IV is added to provide a new analysis based on EFR, which is verified experimentally in Section V-F. We have also extended our method to multibit networks, along with more experimental results using larger datasets and deeper networks in Section V.

TABLE I
CLASSIFICATION AND COMPARISON OF PREVIOUS WORK

Category	Method	Downside
Retrain	Weight significance [17], [18]	Requires dataset
	Threshold train [21]	Requires dataset
	KD (Knowledge Dist.) [19]	Requires dataset
Correct	Output compensation [24], [25]	HW overhead, Extra calc.
	RSA [19]	HW overhead, Extra calc.
	X-ABFT [20]	HW overhead, Extra calc.
Remap	HW redundancy [7]	HW overhead (crossbar)
	Neuron permutation [21], [23]	Inapplicable to conv. layer
	Row permutation [17], [24]	HW overhead (router)
	Row flipping [23]	HW overhead (negation)
	Input splitting [26]	HW overhead (input split)

II. RELATED WORK

A. SAF Mitigation Techniques

Previous work on the SAF mitigation in RCAs can be divided into three categories (see Table I). The first is retraining, which is simply to train the DNN again using a gradient-descent training algorithm while fixing some weight elements to constant values based on SAF information [17], [18], [19], [21]. Retraining can recover the accuracy in the highest level without hardware cost. A retraining approach can employ a very elaborate training scheme in order to maximize fault recovery; for instance, knowledge distillation (KD)-based retraining, which uses a teacher–student model, is demonstrated to help improve fault recovery [19]. Furthermore, retraining does not require any postprocessing, and can be easily combined with postprocessing methods such as remapping (see below). On the other hand, two things are required for retraining, a fault map [8], [27] and a training dataset. Obtaining an exact fault map can be very time consuming. Besides, the computational demand of retraining is high, and retraining must be repeated for each individual DNN chip, adding to the computation complexity.

The second category is correction, which is to correct the output of a faulty ReRAM crossbar array via a postprocessing step [19], [20], [24], [25]. A faulty ReRAM crossbar array can lead to distortion of MVM result. A simple technique was proposed in [25] which can correct this by postprocessing. The contribution of faulty ReRAM devices toward MVM result is calculated in the postprocessing step with full access to the input vector and the faulty ReRAM cells. They observe that it is essentially the same as doing another sparse MVM operation. In order to avoid reliability issues, they implement this postprocessing in CMOS circuit. Although it gives very high accuracy recovery, it has very high cost due to the additional MVM operation, and requires a digital hardware module. The other techniques [19], [24] are very similar to [25], though RSA [19] aims to reduce the overhead of the correction step by exploiting weight importance. One interesting method in this category is the application of algorithm-based fault tolerance (ABFT) to RCAs [20], which is based on checksum vectors and fault-detection signatures. However, this approach still requires hardware redundancy, which is found to be about 33% in the case of [20].

The third category is remapping, which adds a preprocessing step before a faulty ReRAM crossbar array, thereby enhancing the quality of result despite the presence of SAFs. An example of remapping is matrix permutation [17], [21], [23], [24], which is based on the idea that by reshaping or rearranging a weight matrix one can increase the number of *matches*, or the cases where a weight value at a cell location happens to be the same as the fault value at the cell. Matrix permutation can be divided into row permutation (RP) [17], [24], which requires an additional router, and *neuron permutation* [21], [23], which does not require a router. However, matrix permutation requires considerably larger processing time to find the optimal permutation as the network size increases. Furthermore, in the case of neuron permutation, weight positions must be reshaped, which becomes a problem if a convolution layer is performed with RCAs due to the weight sharing property of a convolution filter. Thus, neuron permutation is not feasible in the convolution layers and is limited to fully connected layer only model.

The idea of row flipping (RF) [23] is to flip the sign of an entire row (that has one or more SAF cells) and shift the values of the row to be as close as possible to the SAF value(s). This method is fairly straightforward as it only changes the sign and does not require a time-consuming postprocessing step. However, to recover the correct output, the RF method requires a scalar value that is the sum of an input vector [23], calculation of which requires extra hardware.

There are some techniques to tolerate variation or SAF in ReRAM arrays. Sun et al. [28] proposed a training-free method to overcome variation. They use a new coding scheme where each bit has the same significance, and the encoded value is expressed as the sum of all the bits. However, for BNNs, the new coding scheme is the same as the general binary coding. In other words, this new coding scheme does not give any benefits in BNNs. Our FPT method can work for BNNs as well. Long et al. [29] proposed a variation-aware training methodology where stochastic noise is added intentionally during training to enhance the robustness. Our method, FPT, does not require any training or training dataset if there is a pretrained model, whereas the method in [29] would still require additional training with a training dataset even if a pretrained model is given. Finally, [26] is a calibration method similar to ours. However, it relies on input preprocessing (called input splitting), which must be performed at runtime and therefore requires extra hardware unlike ours.

B. Weight Realization

It is straightforward to implement a weight tensor as resistance matrices of RCAs except that the required size and value range of a weight tensor may exceed those of a RCA. Partitioning [1] can solve the size problem; after that, the output summation of RCAs needs to be done before the activation function. Negative weights can be handled by adding a constant offset so that all the weight values become non-negative. An extra column in RCAs is required to implement the offset, which needs to be subtracted from the RCA output. This is called *unbalanced* weight realization [30]. Alternatively, one can use a pair of RCA arrays (or ReRAM cells), so that signed weight values can be implemented by subtracting the output of one (*negative array*) from that of the other (*positive array*), which is referred to as *balanced* weight realization [1]. For our experiments with BNNs and QNNs, we use the weight realization schemes listed in Table II, where (x, y) in the balanced cases refers to a resistance value pair for positive/negative arrays.

TABLE II
WEIGHT TO RESISTANCE MAPPING. LRS AND HRS REFER TO LOW AND HIGH RESISTANCE STATES, RESPECTIVELY

Scheme	+1	−1
Unbalanced	LRS	HRS
Balanced	(LRS, HRS)	(HRS, LRS)

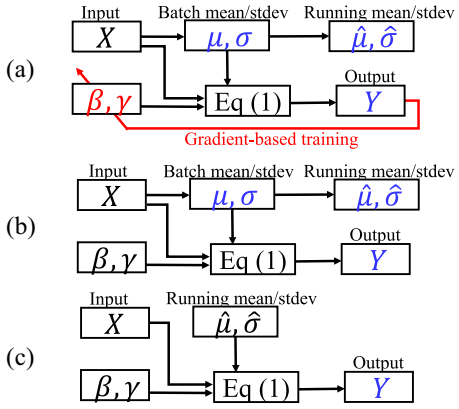


Fig. 1. BN layers run differently during training, FPT, and inference. Variables in red are updated by gradient descent, and those in blue are by the forward phase.

C. Batch Normalization

Batch normalization (BN), which is known to reduce the internal covariate shift problem [31], is essentially an affine transformation. Equation (1) is the formula of a BN layer, where x is the output of the preceding layer which is either a convolution or fully connected layer. A BN layer is widely used in CNN models to improve training performance. For each output channel, it contains four parameters, which are updated during training only. They can be classified into two categories: 1) forward parameters (μ, σ), which are the input statistics per batch, and 2) backward parameters (β, γ). Forward parameters and backward parameters are updated in the forward phase and backward phase, respectively, [see Fig. 1(a)]. For inference, μ and σ are replaced with constant values, which may be obtained from the EMA (exponential moving average) of μ, σ during training [see Fig. 1(c)]. To further simplify computation, BN layers can be folded into the preceding layers, resulting in modified weight/bias values [32]

$$y = \gamma\left(\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}\right)+\beta. \tag{1}$$

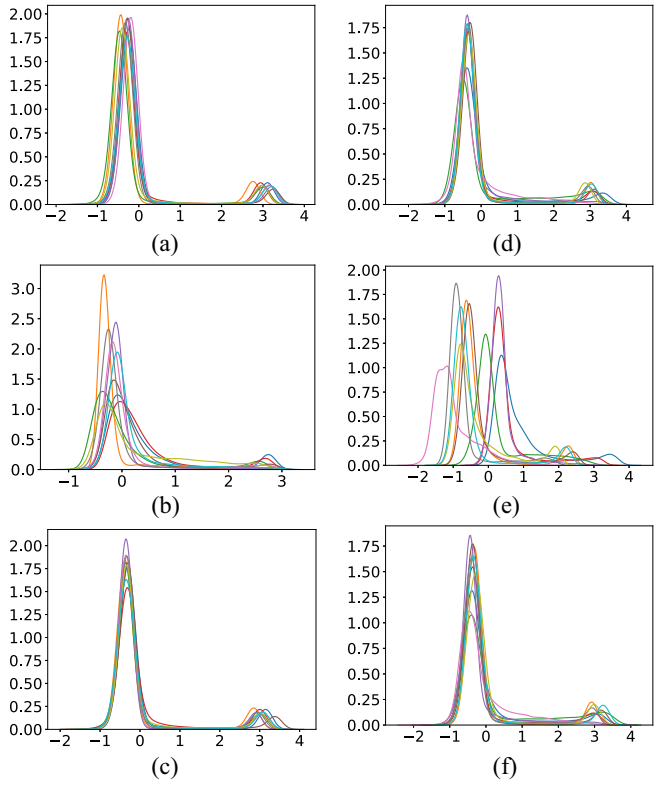


Fig. 2. Distribution of output activation (after BN) at the last layer before soft-max in BNN, where (a)–(c) show the results with the MNIST dataset, and (d)–(f) show the results with the fashion-MNIST dataset. Different colors mean different output neurons (stuck-at-open 9%, stuck-at-close 1%).

III. Forward Parameter Tuning

A. Motivation

Optimization Perspective: Our original motivation comes from the need to reduce cost. Given enough hardware resources, the correction approach can almost achieve 100% accuracy recovery, but the cost is very high. It would be ideal to restore accuracy by changing biases only without explicit gradient backpropagation-based training, which requires a dataset and incurs high computation cost. The fact that a BN layer can be folded into the preceding layer motivates us to use BN layers as an alternative to directly modifying bias values.

Statistical Perspective: To see the effect of device defect, we plot the distribution of output activation with and without SAFs in Fig. 2. Without faults, output distribution is almost identical across neurons, with one large peak around zero and one small peak around 3. With faults, however, we observe varying degrees of distortion among neurons (mean is shifted and standard deviation is increased). Thus, one may expect to see improved accuracy if the distribution graphs are changed back to their original shapes, which is, coincidentally, one of the primary objectives of BN. Indeed, we observe that most of accuracy degradation caused by SAFs can be recovered by just BN training (see Table V). Since BN-only training is still a training, we explore a variation of BN-only training, which we call *Forward Parameter Tuning*, in which we adjust the forward parameters (μ, σ) only. Our FPT method updates forward parameters during the forward phase and does not require back propagation or a full dataset. Therefore, FPT represents a computationally cheap operation that is free of dataset or extra hardware.[3]

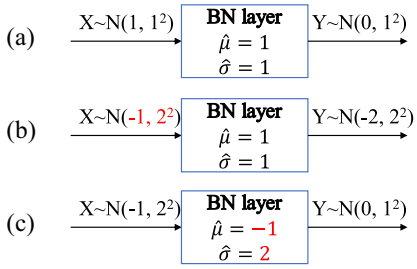


Fig. 3. Example of our FPT method. (a) Without SAF. (b) With SAF. (c) With SAF and FPT.

B. Detailed Method

Fig. 3 illustrates an example showing how our FPT method may help recover the correct output despite SAFs. Here, we assume that the input (X) to a BN layer follows a Gaussian distribution with $\mu = 1$ and $\sigma = 1$. Then, the running mean ($\hat{\mu}$) and standard deviation ($\hat{\sigma}$) parameters of the BN layer would be 1 and 1, respectively, and the output (Y) of the BN layer should follow a Gaussian distribution with $\mu = 0$ and $\sigma = 1$ [see Fig. 3(a)]. Now, with SAFs in the preceding layer, the statistics of X may be changed as illustrated in Fig. 3(b). However, because the BN layer parameters are not updated during inference, the distortion in the input will be reflected in the output as well, ultimately distorting in the final output. In Fig. 3(c), the FPT method updates the BN layer parameters, which can bring back the original, correct distribution of the output data. In reality, however, the effect of SAFs can be more devastating (e.g., data belonging to different categories may be affected differently), which explains why the FPT method may not be enough to recover the original output.

Fig. 4 shows our experimental flows for the BNN and QNN (or multibit weight network) case. For each case, there are two flows. When not using our method, the default flow, labeled *simple inference*, is to first train a neural network, then fold BN layers if it is a BNN, and deploy the network, which is to run inference with the network on ReRAM arrays possibly laden with various nonidealities. The *Inference with Hardware Simulation* step means that network inference is performed while simulating SAFs injected into RCAs. Alternatively, an already trained model is taken, and the mean and std. deviation of BN layers are updated by running a few dozen additional iterations using a calibration dataset. BN layer folding is done only for BNNs, since for a multibit weight network, updating BN layer parameters (μ and σ) means that the weight parameters can change when the BN layer is folded to the preceding layer. But the changed weight parameters may not be programmed correctly due to SAFs. Therefore, for multibit weight networks, we assume BN layers not to be folded but to be implemented in digital hardware in order to guarantee accuracy. For BNNs, on the other hand, folding a BN layer changes only the bias but not the weight parameters. Therefore, we can use BN folding for BNNs.

[3]We do need a calibration set to get the statistics of input, which however can be unlabeled and small.

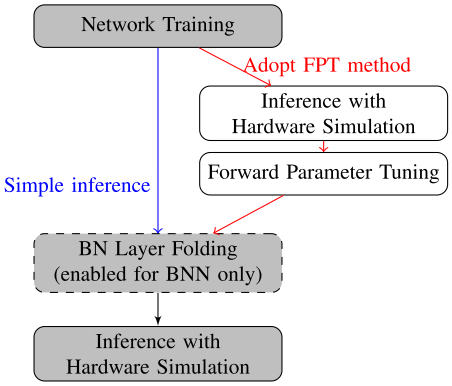


Fig. 4. Our experimental flow (note: BN Layer Folding is enabled only for BNN). After network training, there are two scenarios, which are shown by colored arrows. The blue arrow shows the simple inference scenario. The red arrows show the scenario of adopting our FPT method. In both scenarios, BN layers can be folded in the case of a BNN. Gray blocks are common steps while transparent ones are unique for the FPT method.

The FPT step is not a training step, and is more like inference since there is no backward update. Only the running mean and std. dev. in BN layers are updated during forward pass. In the absence of SAFs, the statistics (i.e., mean and std. dev.) of each batch will be very similar to the (final) running statistics. But the existence of SAFs will change the batch statistics, which is averaged over several iterations using EMA to generate a new running statistics (EMA is initialized to the final running statistics of training). For DNNs without BN layers, new BN layers can be added next to convolution/fully connected layers to mitigate SAF, and can be removed after FPT by BN layer folding [32].

C. Analysis: Whether Adding Affine Transformation Can Improve the Classification Accuracy at All

One might ask how adding a simple operation like BN can help improve accuracy at all. To answer, we model a neuron output and distortion as random variables and see if adding an affine transformation can result in higher classification accuracy. For generality, we assume that the neuron output can have up to two peaks and the distortion can have different distribution parameters depending on the target output.

Let X_0 and X_1 be random variables with Gaussian distribution modeling the output of a neuron, representing two peaks (see Fig. 2)

$$p(x_0) = \mathcal{N}\left(x \mid \mu_{x0}, \sigma_{x0}^2\right) \tag{2}$$

$$p(x_1) = \mathcal{N}\left(x \mid \mu_{x1}, \sigma_{x1}^2\right). \tag{3}$$

The decision boundary θ can be chosen to be the point where the two Gaussian probability density functions (PDFs) meet.

Let us assume that the effect of SAF in the weight parameters can be modeled as Gaussian noise with different parameters depending on the target output

$$G_0 \sim \mathcal{N}\left(\mu_{g0}, \sigma_{g0}^2\right) \tag{4}$$

$$G_1 \sim \mathcal{N}\left(\mu_{g1}, \sigma_{g1}^2\right). \tag{5}$$

Then, $X_0' = X_0 + G_0 \sim \mathcal{N}(\mu_0, \sigma_0^2)$ and $X_1' = X_1 + G_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$, where $\mu_i = \mu_{xi} + \mu_{gi}$ and $\sigma_i^2 = \sigma_{xi}^2 + \sigma_{gi}^2$ with $i = 0, 1$.

We add an affine layer modeling BN as $Y_i = aX_i' + b$, where the computation of BN is assumed to be free of SAF. Now, we can estimate the error probability with the SAF-induced Gaussian noise. Using the previous decision boundary θ, the probability of error with noise and BN applied is given as

$$e_0 = P(Y_0 > \theta) \quad \text{when the true output is 0} \tag{6}$$

$$e_1 = P(Y_1 < \theta) \quad \text{otherwise.} \tag{7}$$

Now, our objective is to find a and b that can minimize the total error rate $\mathcal{L} = \phi_0 e_0 + \phi_1 e_1$, where ϕ_0 and ϕ_1 are the prior probability of the true output ($\phi_0 + \phi_1 = 1$). We take partial derivatives of $\mathcal{L}$ to find a minimum, which gives us the following equation to find the optimal values of a and b:

$$\frac{\phi_0}{\sigma_0} g\left(\frac{\theta - b}{a\sigma_0} - \frac{\mu_0}{\sigma_0}\right) = \frac{\phi_1}{\sigma_1} g\left(\frac{\theta - b}{a\sigma_1} - \frac{\mu_1}{\sigma_1}\right) \tag{8}$$

where g is the standard Gaussian PDF. For a simple case where $\sigma_0 = \sigma_1 = 1$, (8) becomes $\phi_0 g(x - \mu_0) = \phi_1 g(x - \mu_1)$, where $x = (\theta - b)/a$, meaning that x is the point at which the two Gaussians centered at μ_0 and μ_1 meet. Clearly, x will coincide with θ if the two Gaussians (at μ_0 and μ_1) had the same parameters as in (3), i.e., if there were no noise. With noise, (a, b) minimizing $\mathcal{L}$ can be different from $(1, 0)$.

Though finding a closed-form solution to the above equation is difficult, this analysis shows that an affine transformation such as BN can indeed help reduce classification error in the presence of random noise. Note that we only show optimal $(a, b) \neq (1, 0)$ can exist, but we do not claim our FPT method can find the optimal values; our method is only a heuristic.

D. Note on Implementation

BN inference itself [Fig. 1(c)] is very simple, involving just an affine transformation (one multiplication and one addition per output), which can be implemented in digital hardware. We will discuss it in detail in Section V-I.

BN tuning [Fig. 1(b)] updates the statistical parameters in the forward path by running a few dozen additional iterations using a calibration set. This can be done either by software running on a host machine or by additional hardware. Here, we assume that the statistical parameter tuning is done by software running on a host machine. We get the input of each BN layer (the output of each convolution and fully connected layer followed by BN layer) and then send it to the host machine to compute and update the mean and standard deviation. Once the BN layers are tuned with these statistical parameters, we can fold BN layers for BNNs, so that no additional hardware is needed for inference. In multibit weight networks, we use a simple affine transformation as mentioned above to implement each BN layer. Note that the tuning process only requires the input of BN layer, unlike [21] which requires full knowledge of the location and state of the faults, and needs to be performed once unless the fault state changes.

IV. Analysis of Effective Fault Rate

To better understand the effect of FR on computation accuracy, we introduce a new measure called *EFR*. We also present an analytical method to calculate EFR for any given FR under certain assumptions, namely uniform weight distribution and balanced weight mapping. Recall that balanced weight mapping (see Section II-B) maps a weight parameter to a pair of ReRAM devices, which may be arranged in two columns or two arrays, and the correct result can be obtained by subtracting the output of one column (such columns or arrays are referred to as *negative*) from that of the other column (positive).

A. Effective Fault Rate

The term FR is defined as the probability of a cell being stuck at a certain, usually extreme, state, such as LRS or HRS. If a value that is to be written to a device happens to be the same as the stuck-at value of the device, it will not result in any observable error. Therefore, we define *EFR* to be the probability of a device having an observable error due to SAFs. To distinguish the original FR from EFR, we call the former *raw fault rate*. Unlike raw fault rate, EFR considers programmed weight values as well, and can be a more direct predictor of application performance as we show in our experimental results.

To illustrate the idea of EFR, consider the scenario of writing $+1$ to a ReRAM cell that can represent a binary value ($\{+1, -1\}$). Fig. 5a illustrates how $+1$ can be programmed using two ReRAM devices. Suppose that the raw FR is 10% and that the probabilities of SA0 and SA1 are the same (i.e., 5% each). Then, each of the ReRAM devices has three possibilities: 1) No SAF, whose probability is 90%; 2) SA0 with 5% probability; and 3) SA1 with 5% probability. Considering the two devices together, we have nine cases. Among them, the following four cases will have no observable error, because the stuck-at values happen to be the same as the write values: {(No SAF, No SAF), (No SAF, SA0), (SA1, No SAF), (SA1, SA0)}. Therefore, the probability of observing an error is 9.75% ($= 1-(0.9\cdot 0.9 + 0.9\cdot 0.05 + 0.05\cdot 0.9 + 0.05\cdot 0.05)$), which is the EFR. This example shows that EFR can be different from raw FR (lower in this case), and depends on weight distribution.

In the multibit weight case, the likelihood of having no observable error diminishes, resulting in greater EFR for the same raw FR. Consider a ReRAM device with three resistive states that correspond to $\{0, 0.5, 1\}$. Then, with two such devices one can create a cell that can represent any value from a set $\{-1, -0.5, 0, 0.5, 1\}$, which has one more state than a 2-bit integer can represent. Suppose we represent a weight value of 0.5 as shown in Fig. 5b. Then, there are only two cases in which there is no observable error: {(No SAF, No SAF), (No SAF, SA0)}. Therefore, the EFR is 14.5% ($= 1 - (0.9 \cdot 0.9 + 0.9 \cdot 0.05)$), which is higher than that of the binary example. The above examples also show that EFR can be both lower and higher than raw FR depending on device precision.

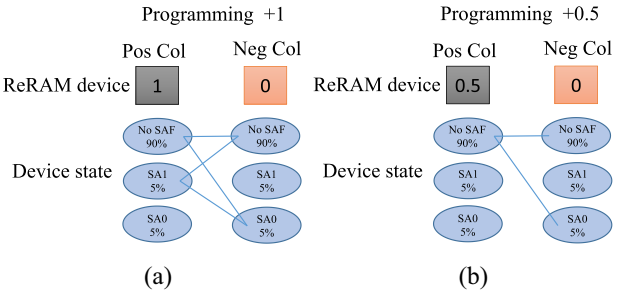


Fig. 5. Example fault analysis (balanced weight realization; raw FR is 10%). (a) In the binary case programming $+1$, four out of nine combinations result in no observed error. (b) In the multibit case programming $+0.5$, only two out of nine combinations result in no observed error.

TABLE III
EFR Formulas (R: Raw FR)

Weight precision	Effective fault rate		
	OCR = 1	OCR = 0.2	OCR = 5
1-bit	$R(1-\frac{R}{4})$	$R(1-\frac{5}{36}R)$	$R(1-\frac{5}{36}R)$
2-bit	$\frac{6}{5}R(1-\frac{R}{3})$	$\frac{22}{15}R(1-\frac{5}{11}R)$	$\frac{14}{15}R(1-\frac{R}{7})$
4-bit	$\frac{24}{17}R(1-\frac{R}{3})$	$\frac{88}{51}R(1-\frac{5}{11}R)$	$\frac{56}{51}R(1-\frac{R}{7})$

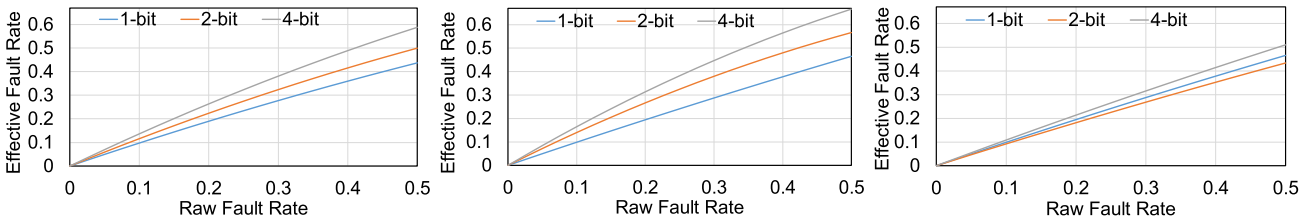
B. Effective Fault Rate Analysis

Using the definition of EFR, we derive EFR formulas for 1 bit ($\{+1, -1\}$ as in BinaryNet [33]) as well as multibit cases. Here, we make one assumption that the weight values are equally distributed; for instance, in the BinaryNet case, the number of $+1$ weight parameters is the same as that of -1 parameters.[4]

The result is summarized in Table III and Fig. 6. Here, OCR means Open-Close-Ratio, the ratio between stuck-at-open faults versus stuck-at-close faults. In other words, if FR $= 10\%$ and OCR $= 4$, we can expect about 8% of ReRAM devices stuck to HRS, regardless of target resistance value, and about 2% stuck to LRS. For OCR $= 1$ and OCR $= 0.2$, this result suggests that for any given raw FR, EFR of 2 bit (or 4 bit) is greater than that of 1-bit (or 2-bit) weight. In other words, EFR increases as weight precision increases, regardless of raw FR. However, for OCR $= 5$ case, EFR of 2 bit is less than that of 1-bit weight. For higher OCR, the EFR of multibit weight can be lower than 1-bit weight, but for most of cases, multibit has higher EFR. As a consequence, it is more difficult to mitigate the effect of SAF in multibit networks by using the FPT method because of the higher EFR.

Our definition of EFR has a limitation that it does not consider *error distance*. For instance, a cell stuck at zero will have an observable fault as long as the cell has a nonzero weight value; however, the error distance will be different if the weight value is 0.5 versus 1.0. Our definition does not

[4]This assumption is to simplify our analytical model. Weight parameters of a multibit network typically follow a bell-shaped distribution (i.e., more values are found around zero), in which case the EFR will be different with the case of uniform distribution.



(a) (b) (c)

Fig. 6. Effective versus raw FR. (a) OCR = 1. (b) OCR = 0.2. (c) OCR = 5.

TABLE IV
BASELINE ACCURACY FOR ALL MODELS

BNN	MNIST	CIFAR-10	CIFAR-100	ReActNet	
	98.00%	91.27%	59.65%	65.96%	
CIFAR-10 QNN	A1W1	A1W2	A2W2	A1W4	A4W4
	90.08%	91.26%	91.83%	91.02%	92.39%

consider such differences, which can limit the accuracy of our EFR formulas in predicting the output distortion due to SAF.

V. EXPERIMENTS

A. Experimental Setup

To evaluate the effectiveness of our proposed method, we use BNNs as well as QNNs. Table IV lists the models used in our experiments as well as their baseline accuracy. For BNN experiments, we use the MNIST, CIFAR-10, CIFAR-100, and ImageNet datasets. The MNIST BNN model, which is from [33], is an multilayer perceptron (MLP), and the CIFAR-10 BNN, which is also from [33], is a VGGNet [34] with 3× inflation factor, and the CIFAR-100 BNN is the ResNet20 from [33]. For ImageNet, we use the ReActNet model [35].

For QNN experiments, we use the CIFAR-10 dataset and a VGGNet model with 1× inflation factor. We denote the activation precision x and weight precision y of a QNN model as AxWy, where 1 bit means $\{-1, +1\}$ instead of $\{0, 1\}$, 2 bit includes five states ($\{-1, -0.5, 0, +0.5, +1\}$), and similarly, 4 bit includes 17 states.

Our experimental flow (Fig. 4) is implemented in PyTorch, and supports weight partitioning and mapping to RCAs as well as our FPT method as described in Fig. 4. BNN models are trained using the training procedure in [33]. The QNN models are trained using progressive fine-tuning [36], which provides better initialization. That is, we first train the floating-point model, and a higher precision model using the floating-point model as the initial weight, and a lower precision model using a higher precision model as the initial weight, and so on.

In this work, we consider a multibit device [37], which has resistance range of 1 kΩ–1 MΩ with linear voltage-conductance relation and up to 5-bit precision. For SAF injection we follow the methodology in [25].

For the calibration set, we use a randomly selected subset of the training dataset. We vary FR, which is raw FR, and use OCR values of 5, 1, 1/5.[5] Our baseline accuracy is the test accuracy of each BNN or QNN model without any nonideality.

[5]Previous work uses various OCR values including 0.225 [18] and 5.1 [8].

TABLE V
COMPARING VARIOUS TRAINING METHODS AND OUR FPT FOR MNIST BNN (UNBALANCED CASE, OCR = 1, 10 EPOCHS). SI REFERS TO SIMPLE INFERENCE WITHOUT RETRAINING

MNIST test accuracy					
FR	SI	FPT	Bias train	BN train	Retrain
10%	97.36%	97.47%	97.81%	97.93%	97.32%
20%	91.85%	97.21%	97.23%	97.35%	96.92%
40%	44.23%	95.07%	94.93%	96.7%	97.19%
CIFAR-10 test accuracy					
FR	SI	FPT	Bias train	BN train	Retrain
10%	78.18%	88.83%	89.25%	90.97%	91.40%
20%	32.46%	86.08%	87.21%	89.29%	90.76%
40%	10.08%	66.30%	68.18%	82.39%	87.75%

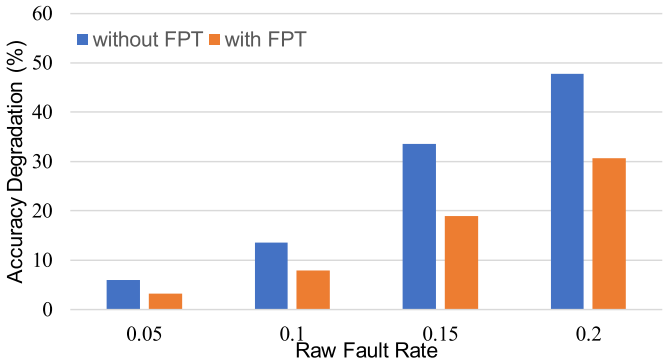


Fig. 7. ResNet20-CIFAR100 results (Binary, OCR = 1, balanced). y-axis is the accuracy degradation from the binary baseline (without SAF).

B. Effectiveness of Our FPT Method

Table V compares various partial retraining/full retraining methods with our FPT method and simple inference (as defined in Fig. 4). BN training updates backward parameters (β, γ) of BN using backpropagation, but weight parameters of convolution/fully connected layers are not touched. Bias training updates the bias of each convolution/fully connected layer only. Up to FR 20%, our FPT result is comparable to that of other training methods; only 3% drop is observed at FR 40% in MNIST. This result suggests that our FPT method can closely follow (partial) retraining methods.

Fig. 7 shows the ResNet20 result on the CIFAR-100 dataset and Table VI shows the ReActNet result on the ImageNet dataset. These results show that our FPT method can recover nearly half of the accuracy degradation, demonstrating that our FPT method can be useful for deeper neural networks and larger datasets as well.

When we have a large FR such as 30%, the model capacity is reduced to 70% which leads to significant drop in the

TABLE VI
PERFORMANCE OF FPT ON IMAGENET (REACTNET, OCR = 1, BALANCED). NOTE HERE THAT SAF IS NOT INJECTED TO THE FIRST AND THE LAST LAYER

FR	Inference only	FPT
8%	47.27%	56.09%
10%	36.23%	54.26%
12%	24.62%	50.05%
14%	13.67%	45.56%

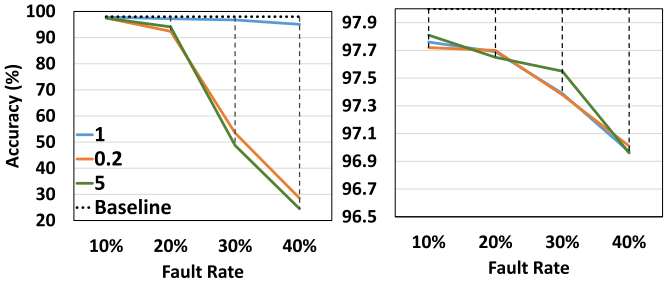


Fig. 8. Comparison between (a) unbalanced and (b) balanced realizations of the weights (MNIST BNN). Different colors mean different OCR values.

TABLE VII
PERFORMANCE OF FPT WHEN USING TRAIN VERSUS TEST DATASET AS THE CALIBRATION DATASET (CIFAR-10 BNN, OCR = 1, UNBALANCED)

FR	Train dataset	Test dataset
10%	89.74	88.04
20%	87.52	85.36
40%	67.89	67.37

baseline model accuracy. Our FPT method does not change the weight. It only fine-tunes the statistics parameters in the BN layer, so it is hard to recover the accuracy at a higher FR, which is a potential limitation of our approach.

C. Effect of OCR and Weight Realization

In this section we explore several cases of OCR to explore more realistic ReRAM SAF cases, the result of which is summarized in Fig. 8a. The OCR values of 5 and 1/5 are chosen based on [8] and [18]. The graph shows that our FPT method is sensitive to OCR value for unbalanced weight realization, with OCR = 1 being the best. On the other hand, using balanced realization (see Fig. 8b) gives great fault recovery regardless of OCR values. We observe that even the most skewed OCR values result in quite small accuracy drop of up to 1.04% when FR = 40%.

D. Calibration Dataset

In constructing the calibration dataset, we need to know from where to get data and how many images to use. Table VII compares the training versus test dataset as the source of our calibration set. For this experiment, we use the maximum size, i.e., 10 000 images, which is the size of the test dataset. Overall, the training set is slightly better than the test set, which may suggest that using independent data (i.e., data that is not used during inference) causes no performance degradation.

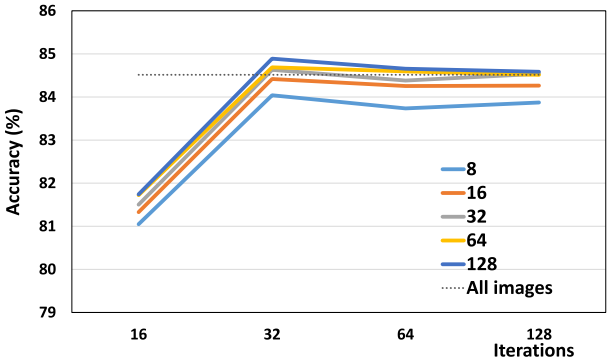


Fig. 9. Performance versus the number of iterations for FPT, with varied batch size (CIFAR-10 BNN, Unbalanced, FR = 10%, OCR = 1/9). Different colors mean different batch sizes.

TABLE VIII
COMPARISON WITH OTHER MITIGATION METHODS. CIFAR-10 BNN TEST ACCURACY (OCR = 1/5, BALANCED). SI REFERS TO SIMPLE INFERENCE

FR	FPT	SI	RF	RP	RF+RP
10%	No	82.32%	87.85%	89.80%	89.81%
	Yes	88.53%	89.51%	89.37%	89.56%
20%	No	41.21%	57.46%	78.86%	76.38%
	Yes	88.08%	88.37%	88.81%	89.08%

Fig. 9 provides an answer to the minimum number of images for the calibration set. We use unbalanced, OCR = 1/9 for the worst case simulation. The graph shows the number of iterations used in FPT by specific batch sizes. The calibration dataset size is the product of the number of iterations and the batch size. Each graph is the average accuracy of ten independent experiments. The graph clearly shows that the accuracy has stronger correlation with the number of iterations, rather than the batch size. Overall, 1024 images (32 iterations, batch size 32) shows only 0.5% drop from the best case, and thus can be recommended as the minimum calibration set size. In the case where there are not enough images, one guideline is to divide the available data into 32 iterations, possibly with a rather small batch size.

E. Comparison With Previous Methods

We focus on the comparison with the remapping methods, and among them, especially RF [23] and RP [17], since they can be applied to any network with relatively low cost (especially compared with the correction methods). Table VIII summarizes the result of applying various combinations of RF and RP, with and without our FPT method. At 10% FR, all mitigation methods tend to show quite high accuracy recovery. However, at 20% FR, the performance of the previous methods starts to deteriorate very badly, that is, unless the FPT method is used together. While FPT alone gives a similar result as applying RF and RP together with FPT (only 1% difference), between the two, RF may be more cost effective to apply together due to the differences in the hardware cost and implementation complexity of RF and RP.

F. Accuracy of Our Analytical Formulas

To validate the accuracy of our analytical formulas for EFR, we compare our formula's predictions with simulation results.

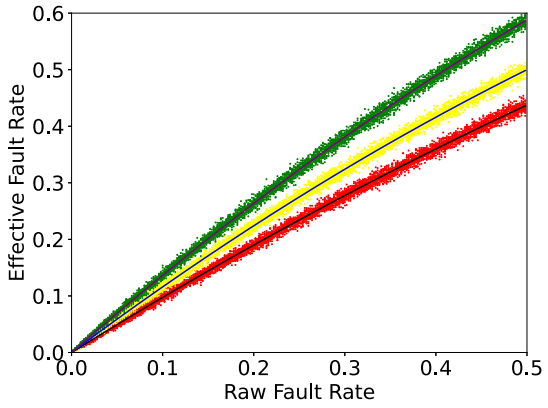


Fig. 10. EFR formulas (curves) versus simulation results (points). Balanced weight realization (OCR = 1). Three curves are 1, 2, and 4-bit cases from bottom to top.

Our fault injection simulation is performed as follows. For each raw FR, we generate SAFs randomly for each given raw FR 25 times, adding SAFs to weight values. For weight values, we use a 4-D tensor of quantized weight (the shape of which is $170 \times 3 \times 3 \times 3$) generated according to a uniform distribution. The weight tensor is generated once and reused in all the experiments. Fig. 10 shows the result, which confirms that our formulas are correct. We have repeated the same experiment for the other values of OCR, and their results are also very similar (not shown).

G. Results for Multibit Networks

Fig. 11 shows the effect of FPT on CIFAR-10 QNN test accuracy. The results suggest that our FPT can recover the baseline accuracy, which is the accuracy without any SAF, when raw FR is around 20% or less. However, when FR is over 20%, the A1W2, A1W4, and A4W4 cases start to show much lower accuracy than the baseline. Also, there is a gap between the binary (A1W1) case and the multibit cases, which is increasing as FR increases.

To better understand the differences between the binary versus multibit cases, Fig. 12(a) and (b) plots accuracy as a function of EFR for different weight precision cases (inputs are all binary). First, the results suggest that there is greater accuracy degradation with the multibit weight cases than with the binary case. This trend is common regardless of whether FPT is used. The larger accuracy drop with QNNs seems to be a side effect of highly tuned networks; more precise weights afford the network a higher level of tuning capability, but the degradation is also greater when weights are damaged by imperfections.

Second, all three networks show similar accuracy when FPT is enabled, with the exception of the OCR-5 case, which suggests that our FPT is at least as effective with QNNs as with BNNs. Third, in the case of the OCR-5 case, accuracy degradation is so steep with QNNs that even after FPT there is still a large accuracy gap between BNNs and QNNs. These graphs suggest that the effectiveness of FPT can be seen as reducing the EFR by 3–4 times; that is to say, what may be achieved at EFR = 10% without FPT can be achieved with FPT at EFR = 30%–40%. However, the exact contribution of FPT in terms of EFR reduction is difficult to quantify due to the large degree of variation of such experimental results.

Finally, a comparison between OCR = 1 case in Figs. 12(b) and 13, which plot the same data using differently scaled x-axes, reveals that EFR is indeed a better predictor of network accuracy, demonstrating that EFR can be a useful tool for predicting the accuracy of faulty neural networks.

To understand the reason for QNN's particular underperformance in the case of OCR-5, we compare weight distributions with and without SAF in Fig. 14. The weights here are the quantized weights in the last fully connected layer of CIFAR-10 QNNs. The graphs show that one effect of SAF is to distort the weight distribution so that even though the number (or probability density) of near-zero weights is not particularly high in the case of "no SAF," it becomes so when there is SAF. Moreover, in multibit case, the number of near-zero weights is disproportionately large when OCR is 5, severely impacting the network accuracy, regardless of whether FPT is used.

H. Effect of Other RCA Nonidealities

ReRAM has many other nonidealities, such as variability and IR drop [9], [10]. Variability is considered as one of the major problems for ReRAM especially when compared to other emerging technologies such as STT-RAM. Variability problem may occur during either device programming, reading or both. We have tested the effect of FPT in the presence of both SAF and variability in binary ReRAM.

IR drop effect is simulated by a SPICE-equivalent simulator, presented in [38]. Variability is modeled as additive Gaussian noise to RCA resistance as follows: $R' = R(1+\epsilon)$, where $\epsilon \sim N(0, \sigma^2)$. We vary σ from 0.1 to 0.5. We use the following device parameters: $R_{\text{HRS}} = 1$ MΩ, $R_{\text{LRS}} = 1$ kΩ, and crossbar size $= 64 \times 64$.

Fig. 15a shows the result, where variability is modeled as the Gaussian noise to ReRAM resistance values.

The graph shows that accuracy drop steadily increases as variability increases. The graph also suggests that the impact of variability depends on the FR; the higher the FR, the greater the impact of variability, which is not surprising. Between device variability and SAF, our result indicates that SAF may be more critical. For instance, while the variability of $\sigma = 0.5$ is quite manageable up to FR = 20% (<10% drop), FR = 40% makes RCAs quite unacceptable for DNN accelerators regardless of variability. In the case of IR drop (see Fig. 15), accuracy drop is generally smaller than that in the variability case, though it has a strong dependence on r_w. Again, our FPT method shows very high resilience, and even up to 40%, we observe only about 6% drop from the baseline accuracy when wire resistance is low ($r_w = 0.1$). But in the case of high wire resistance ($r_w = 1$), FPT shows about 5% accuracy drop up to FR = 20%, but then a very large drop at FR = 40%, which cannot be tolerated without retraining.

I. Hardware Overhead Estimation

As we mentioned in Section III-D, BN tuning is done in software. To estimate the tuning cost, we test the time cost

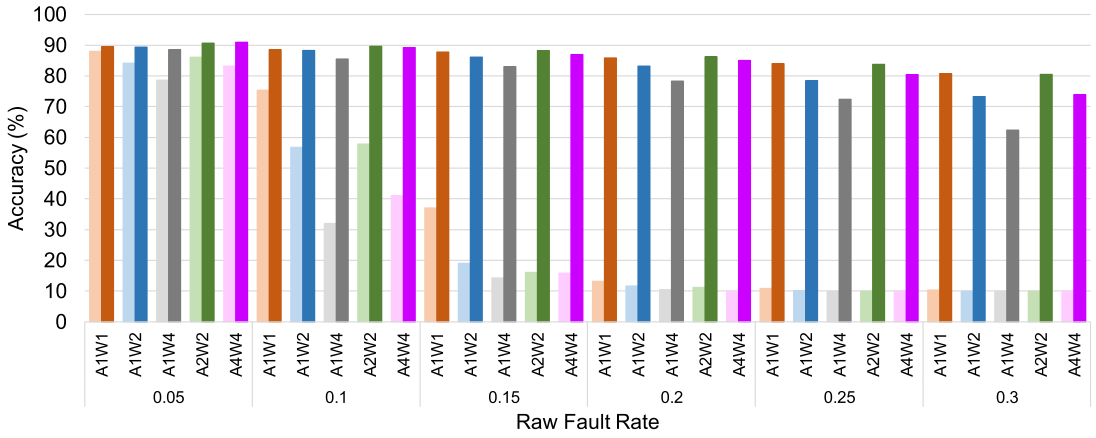


Fig. 11. CIFAR10-QNN results (OCR = 1, balanced). Different colors are different models. Light colored bars are without FPT, and dark-colored bars with FPT.

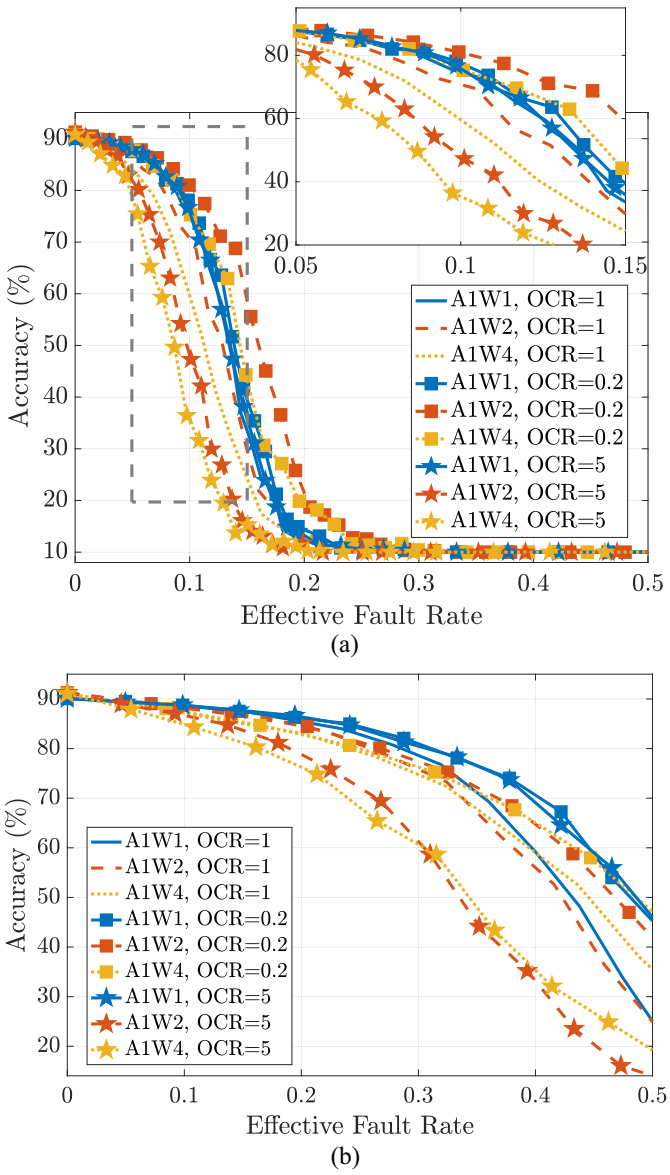


Fig. 12. Accuracy versus EFR. (a) Before FPT. (b) After FPT.

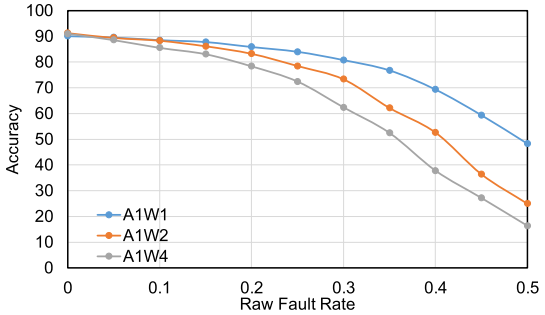


Fig. 13. Accuracy versus raw FR (with FPT, OCR = 1).

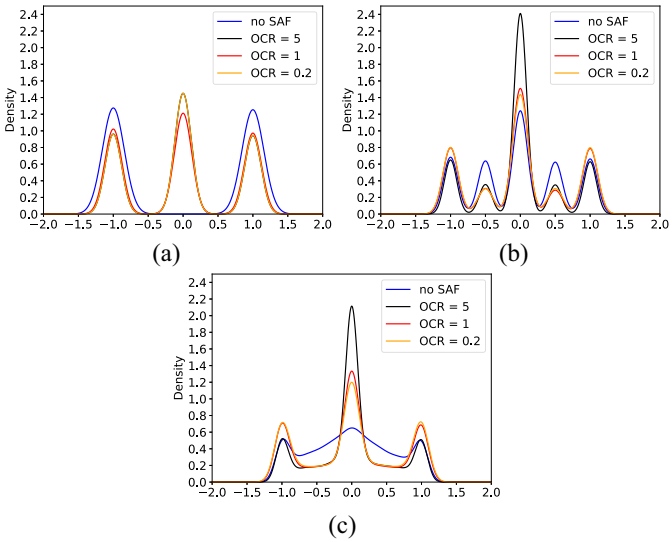


Fig. 14. Weight distributions of different OCR. (a) 1-bit case. (b) 2-bit case. (c) 4-bit case.

per iteration for different approaches as shown in Table IX. The result shows that our FPT method has a much lower cost compared to other training approaches. The time per iteration of our technique is around 2.48 times smaller than other training techniques since it does not require backpropagation. The total time cost of our approach is 152.07 times smaller than others. The main reason is that our FPT method requires much fewer iterations as discussed in Section V-D. Moreover, the calibration set for FPT can be much smaller than the training set as observed in Section V-D. Note that we compare against these techniques on GPU which is not the case for hardware, so the cost of all the training techniques is close. However,

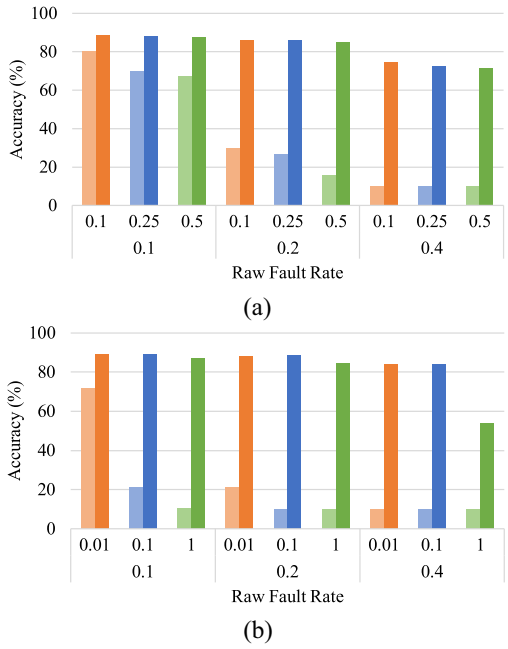


Fig. 15. Nonidealities experiment on CIFAR-10 BNN (OCR = 0.2, balanced). The x-axis shows the standard deviation (σ) of variability [top row in (a)] and the wire resistance [top row in (b)], as well as the raw FR (bottom row). Light colored bars are without FPT (i.e., baseline), and dark-colored ones with FPT. (a) Variability result. (b) IR drop result.

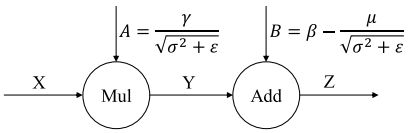


Fig. 16. BN hardware flow.

TABLE IX
TIME COMPARISON WITH OTHER TRAINING METHODS. (CIFAR-10, BNN, BATCH SIZE=256) WE ASSUME TEN EPOCHS OF FINE-TUNING FOR THE RETRAINING METHODS

Method	FPT	Bias train	BN train	Retrain
Average Time (s/*iteration*)	0.029	0.072	0.072	0.072
Total Time (s)	0.928	141.12	141.12	141.12

practically, retraining methods need to rewrite the resistance in RCAs. Besides, for all the training techniques, it is mandatory to get precise fault information from the hardware, which is cumbersome and costly because a typical accelerator may include hundreds of RCAs or more. Therefore, the training techniques could take more time in reality.

During inference, a BN layer is just an affine transformation, which can be efficiently implemented in digital hardware. Fig. 16 shows the flow of our BN hardware. According to (1), BN can be implemented as a multiplication operation and an addition operation, where X is the input activation to BN, and $A = (\gamma/[\sqrt{\sigma^2+\epsilon}])$, $B = \beta - (\mu/[\sqrt{\sigma^2+\epsilon}])$. Note that the statistical parameters are updated by FPT in software, and the BN hardware only uses the parameters. X, A, and B are all fixed-point values, X and A are 8 bit, and B is 16 bit. We have designed BN hardware in Verilog HDL, which is synthesized with a Synopsys Design Compiler using Samsung 65 nm technology. The estimated power dissipation is 3.62 mW with input size of 64 at 100 MHz. Using the RCA power consumption data from [1], we estimate the power consumption of BN to be around 2.56% of the total power, which is low. Moreover, [1] uses 32-nm technology, which is not available for us. If we scale our BN implementation to 32-nm technology, the BN hardware overhead would be much lower.

VI. CONCLUSION

In this article, we presented a novel method to mitigate the effect of permanent faults in RCAs. Our method does not require additional hardware or large datasets, but only updates mean and variance of the BN layers using a tiny fraction of unlabeled data. BN layers are folded back into preceding layers, thus incurring no additional computation either. In addition our method is orthogonal to remapping techniques as demonstrated by our experimental results, and shows a similar level of fault recovery as backpropagation-based training methods, up to about 20% FR. We have also applied FPT to multibit networks and developed the EFR measure to analyze the relationship between accuracy loss and FR. Our experimental results demonstrated that our FPT method works equally well on multibit networks as well as the efficacy of our EFR as a tool for predicting the accuracy of faulty RCA-based neural networks. Our FPT method also demonstrates good harmony with techniques for other nonidealites, including variability and IR drop.

From our experimental results, it seems that FPT can also help mitigate the effect of other nonidealities, such as variability and IR drop, but quantifying its effectiveness requires a more careful study and evaluation. Finally, analogue or continuous-valued ReRAM devices can be more useful as they provide higher integration density. Extending our technique to such cases is left for future work.

REFERENCES

[1] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.
[2] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, May 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14441
[3] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2017, pp. 541–552.
[4] S. Lee et al., "Architecture-accuracy co-optimization of ReRAM-based low-cost neural network processor," in *Proc. 30th ACM Great Lakes Symp. VLSI (GLSVLSI)*, Sep. 2020, pp. 427–432.
[5] K. Smagulova, M. E. Fouda, F. Kurdahi, K. Salama, and A. Eltawil, "Resistive neural hardware accelerators," 2021, *arXiv:2109.03934*.
[6] A. Azamat, F. Asim, and J. Lee, "Quarry: Quantization-based ADC reduction for ReRAM-based deep neural network accelerators," in *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, Nov. 2021, pp. 1–7.
[7] L. Xia et al., "Stuck-at fault tolerance in RRAM computing systems," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 1, pp. 102–115, Mar. 2018.
[8] C.-Y. Chen et al., "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 180–190, Jan. 2015.
[9] S. Lee, M. Fouda, J. Lee, A. Eltawil, and F. Kurdahi, "Learning to predict IR drop with effective training for ReRAM-based neural network hardware," in *Proc. DAC*, Jul. 2020, pp. 1–6.

[10] S. Lee, M. E. Fouda, J. Lee, A. Eltawil, and F. Kurdahi, "Offline training-based mitigation of IR drop for ReRAM-based deep neural network accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, May 23, 2022, doi: 10.1109/TCAD.2022.3177002.
[11] M. E. Fouda, S. Lee, J. Lee, A. Eltawil, and F. Kurdahi, "Mask technique for fast and efficient training of binary resistive crossbar arrays," *IEEE Trans. Nanotechnol.*, vol. 18, pp. 704–716, Jul. 2019.
[12] M. E. Fouda, S. Lee, J. Lee, G. H. Kim, F. Kurdahi, and A. M. Eltawi, "IR-QNN framework: An IR drop-aware offline training of quantized crossbar arrays," *IEEE Access*, vol. 8, pp. 228392–228408, 2020.
[13] J. Kim, C. Lee, J. Kim, Y. Kim, C. S. Hwang, and K. Choi, "VCAM: Variation compensation through activation matching for analog binarized neural networks," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des. (ISLPED)*, 2019, pp. 1–6.
[14] S. Lee, M. Fouda, J. Lee, A. Eltawil, and F. Kurdahi, "Fast and low-cost mitigation of ReRAM variability for deep learning applications," in *Proc. IEEE 39th Int. Conf. Comput. Des. (ICCD)*, 2021, pp. 269–276.
[15] C. Huang, N. Xu, K. Qiu, Y. Zhu, D. Ma, and L. Fang, "Efficient and optimized methods for alleviating the impacts of IR-drop and fault in RRAM based neural computing systems," *IEEE J. Electron Devices Soc.*, vol. 9, pp. 645–652, 2021.
[16] J.-Y. Hu, K.-W. Hou, C.-Y. Lo, Y.-F. Chou, and C.-W. Wu, "RRAM-based neuromorphic hardware reliability improvement by self-healing and error correction," in *Proc. IEEE Int. Test Conf. Asia (ITC-Asia)*, 2018, pp. 19–24.
[17] L. Chen et al., "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Proc. Des. Autom. Test Europe Conf. Exhibit. (DATE)*, Mar. 2017, pp. 19–24.
[18] C. Liu, M. Hu, J. P. Strachan, and H. H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *Proc. 54th Annu. Des. Autom. Conf.*, Jun. 2017, pp. 1–6.
[19] G. Charan, A. Mohanty, X. Du, G. Krishnan, R. V. Joshi, and Y. Cao, "Accurate inference with inaccurate RRAM devices: A joint algorithm-design solution," *IEEE J. Explor. Solid-State Computat. Devices Circuits*, vol. 6, no. 1, pp. 27–35, Jun. 2020.
[20] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Algorithmic fault detection for RRAM-based matrix operations," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 3, pp. 1–31, May 2020. [Online]. Available: https://doi.org/10.1145/3386360
[21] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training enabled by on-line fault detection for RRAM-based neural computing systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1611–1624, Sep. 2019. [Online]. Available: https://doi.org/10.1109/TCAD.2018.2855145
[22] G. Jung, M. Fouda, S. Lee, J. Lee, A. Eltawil, and F. Kurdahi, "Cost- and dataset-free stuck-at fault mitigation for ReRAM-based deep learning accelerators," in *Proc. Des. Autom. Test Europe Conf. Exhibit. (DATE)*, 2021, pp. 1733–1738.
[23] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Handling stuck-at-faults in memristor crossbar arrays using matrix transformations," in *Proc. 24th Asia South Pac. Des. Autom. Conf.*, Jan. 2019, pp. 438–443.
[24] F. Zhang and M. Hu, "Defects mitigation in resistive crossbars for analog vector matrix multiplication," in *Proc. 25th Asia South Pac. Des. Autom. Conf. (ASP-DAC)*, 2020, pp. 187–192.
[25] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," in *Proc. 56th Annu. Des. Autom. Conf.*, Jun. 2019, pp. 1–6.
[26] S.-Y. Sun et al., "Cases study of inputs split based calibration method for RRAM crossbar," *IEEE Access*, vol. 7, pp. 141792–141800, 2019.
[27] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Fault tolerance in neuromorphic computing systems," in *Proc. 24th Asia South Pac. Des. Autom. Conf.*, Jan. 2019, pp. 216–223.
[28] Y. Sun et al., "Unary coding and variation-aware optimal mapping scheme for reliable ReRAM-based neuromorphic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 12, pp. 2495–2507, Dec. 2021.
[29] Y. Long, X. She, and S. Mukhopadhyay, "Design of reliable DNN accelerator with un-reliable ReRAM," in *Proc. Des. Autom. Test Europe Conf. Exhibit. (DATE)*, 2019, pp. 1769–1774.
[30] C.-C. Chang et al., "Mitigating asymmetric nonlinear weight update effects in hardware neural network based on analog resistive synapse," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 1, pp. 116–124, Mar. 2018.
[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn. Vol. 37*, 2015, pp. 448–456.
[32] H. Yonekawa and H. Nakahara, "On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2017, pp. 98–105.
[33] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016, *arXiv:1602.02830*.
[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015, pp. 1–14.
[35] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, "ReActNet: Towards precise binary neural network with generalized activation functions," 2020, *arXiv:2003.03488*.
[36] S. Jung et al., "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4345–4354.
[37] M. Hu et al., "Memristor-based analog computation and neural network classification with a dot product engine," *Adv. Mater.*, vol. 30, no. 9, 2018, Art. no. 1705914.
[38] M. E. Fouda, A. M. Eltawil, and F. Kurdahi, "Modeling and analysis of passive switching crossbar arrays," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 270–282, Jan. 2018.

Chenghao Quan received the B.E. degree in measurement and control technology and instrumentation from the Qingdao University of Science and Technology, Qingdao, China, in 2020. He is currently pursuing the combined master's–Ph.D. degree in electrical engineering from the Ulsan National Institute of Science and Technology, Ulsan, South Korea.

His current research interests include efficient deep learning and hardware-aware deep learning.

Mohammed E. Fouda (Senior Member, IEEE) received the B.Sc. degree (Hons.) in electronics and communications engineering and the M.Sc. degree in engineering mathematics from the Faculty of Engineering, Cairo University, Cairo, Egypt, in 2011 and 2014, respectively, and the Ph.D. degree from the University of California at Irvine, Irvine, CA, USA, in 2020.

He worked as an Assistant Researcher with the University of California at Irvine from April 2020 to March 2022. He is currently a Senior Research Scientist with Rain Neuromorphics Inc., Redwood City, CA, USA. He has published more than 150 peer-reviewed journal and conference papers, one Springer book, and three book chapters. His H-index is 26, and he has been cited more than 2400 times. His research interests include analog AI hardware, neuromorphic circuits and systems, brain-inspired computing, memristive circuit theory, fractional circuits and systems, and analog circuits.

Dr. Fouda was the recipient of the Best Paper Award in ICM for 2013 and 2020 and the Broadcom Foundation Fellowship for 2016/2017. He also serves as an associate editor for many journals in addition to serving as a technical program committee member for many conferences. He serves as a peer reviewer for many prestigious journals and conferences.

Sugil Lee received the B.S. degree in mathematical science from the Korea Advanced Institute of Science and Technology, Deajeon, South Korea, in 2017, and the M.S. degree in computer science and engineering from the Ulsan National Institute of Science and Technology, Ulsan, South Korea, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Electrical Engineering.

His current research interests include HW-aware deep neural network training and their energy-efficient implementation methodologies.

Giju Jung received the B.Sc. degree in computer engineering and the M.Sc. degree in biotechnology from the Ulsan National Institute of Science and Technology, Ulsan, South Korea, in 2018 and 2021, respectively.

After graduation, he joined 3View.Com Inc., Suwon, South Korea, as an Engineer specialized in AI. His research interests include neuromorphic computing systems and efficient deep learning.

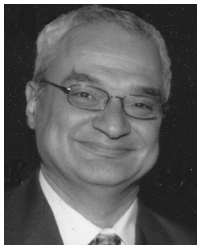
Jongeun Lee (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 1997, 1999, and 2004, respectively.

He has been with the Faculty, Ulsan National Institute of Science and Technology, Ulsan, South Korea, since 2009, where he is a Professor of Electrical Engineering. His research interests include neural network processors, reconfigurable architectures, in-memory computing, and compilers.

Ahmed E. Eltawil (Senior Member, IEEE) received the B.Sc. (Hons.) and M.Sc. degrees from Cairo University, Giza, Egypt, in 1999 and 1997, respectively, and the Doctoral degree from the University of California at Los Angeles, Los Angeles, CA, USA, in 2003.

He is a Professor of Electrical and Computer Engineering with The King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia, where he joined the Computer, Electrical and Mathematical Science and Engineering Division in 2019. Prior to that, he was a Professor of Electrical Engineering and Computer Science with the University of California at Irvine, Irvine, CA, USA, in 2005. At KAUST, he is the Founder and the Director of the Communication and Computing Systems Laboratory. His current research interests are in the general area of mobile communication and computing platforms, with an emphasis on spectrally efficient, low power design.

Dr. Eltawil is the recipient of two certificates of recognition from the United States Congress acknowledging his contributions in the area of wireless communication technologies. He received several awards, including the NSF CAREER Grant supporting his research in low power computing and communication systems. He was selected as the "Innovator of the Year" by the Henry Samueli School of Engineering at the University of California at Irvine, in 2021. He has been on the technical program committees and steering committees for numerous workshops, symposia, and conferences in the areas of low power computing and wireless communication system design. He is a Senior Member of the National Academy of Inventors, USA.



Fadi Kurdahi (Fellow, IEEE) received the B.E. degree in electrical engineering from the American University of Beirut, Beirut, Lebanon, in 1981, and the Ph.D. degree from the University of Southern California, Los Angeles, CA, USA, in 1987.

He has been a Faculty Member with the Department of Electrical Engineering and Computer Science, University of California at Irvine, Irvine, CA, USA, since then, where he conducts research in the areas of computer-aided design, high-level synthesis, and design methodology of large-scale systems, and serves as the Director of the Center for Embedded & Cyber-physical Systems, comprised of world-class researchers in the general area of embedded and cyber–physical systems.

Dr. Kurdahi received the Best Paper Award of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION in 2002, the Best Paper Award in 2006 at ISQED, and four other Distinguished Paper Awards at DAC, EuroDAC, ASP-DAC, and ISQED, and also the Distinguished Alumnus Award from his Alma Mater, the American University of Beirut, in 2008. He was the Program Chair or the General Chair of program committees of several workshops, symposia, and conferences in the area of CAD, VLSI, and system design. He served for numerous editorial boards. He is a Fellow of AAAS.