

# Computing-In-Memory Neural Network Accelerators for Safety-Critical Systems: Can Small Device Variations Be Disastrous?

Zheyu Yan

zyan2@nd.edu

University of Notre Dame

Xiaobo Sharon Hu

shu@nd.edu

University of Notre Dame

Yiyu Shi

yshi4@nd.edu

University of Notre Dame

## ABSTRACT

Computing-in-Memory (CiM) architectures based on emerging non-volatile memory (NVM) devices have demonstrated great potential for deep neural network (DNN) acceleration thanks to their high energy efficiency. However, NVM devices suffer from various non-idealities, especially device-to-device variations due to fabrication defects and cycle-to-cycle variations due to the stochastic behavior of devices. As such, the DNN weights actually mapped to NVM devices could deviate significantly from the expected values, leading to large performance degradation. To address this issue, most existing works focus on maximizing average performance under device variations. This objective would work well for general-purpose scenarios. But for safety-critical applications, the worst-case performance must also be considered. Unfortunately, this has been rarely explored in the literature. In this work, we formulate the problem of determining the worst-case performance of CiM DNN accelerators under the impact of device variations. We further propose a method to effectively find the specific combination of device variation in the high-dimensional space that leads to the worst-case performance. We find that even with very small device variations, the accuracy of a DNN can drop drastically, causing concerns when deploying CiM accelerators in safety-critical applications. Finally, we show that surprisingly none of the existing methods used to enhance average DNN performance in CiM accelerators are very effective when extended to enhance the worst-case performance, and further research down the road is needed to address this problem.

## 1 INTRODUCTIONS

Deep Neural Networks (DNNs) have reached superhuman performance in a variety of perception tasks including speech recognition, object detection, and image classification [19, 20, 32]. Thus, there is an obvious trend to use DNNs to empower edge applications in smart sensors, smartphones, automobiles, and etc. [22, 25, 38] However, because of the constrained computation resources and limited power budget of edge platforms, CPUs or GPUs are not always good candidate computing units for implementing computation-intensive DNNs on edge devices.

Computing-in-Memory (CiM) DNN accelerators [24] is a great alternative candidate for edge DNN implementation because they

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9217-4/22/10...\$15.00

<https://doi.org/10.1145/3508352.3549360>

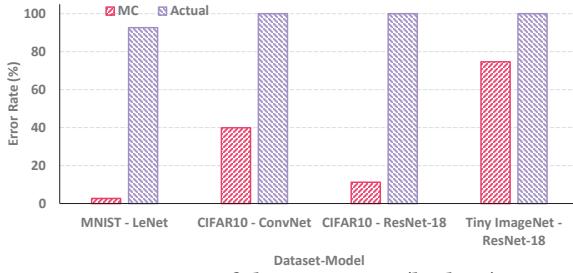
can reduce data movement by performing in-situ weight data access [29]. Furthermore, by employing non-volatile memory (NVM) devices (e.g., ferroelectric field-effect transistors (FeFETs), resistive random-access memories (RRAMs), and phase-change memories (PCMs)), CiM can achieve higher memory density and higher energy efficiency compared with conventional MOSFET-based designs [3]. However, NVM devices can be unreliable, especially because of device-to-device variations due to fabrication defects and cycle-to-cycle variations due to the stochastic behavior of devices. If not handled properly, the weight values provided by the NVM devices during computations could deviate significantly from the expected values, leading to great performance degradation.

To quantify the robustness of CiM DNN accelerators, a Monte Carlo (MC) simulation-based evaluation process is often adopted [23]. A device variation model is extracted from physical measurements. Then in each MC run, one instance of each device is sampled from the variation model and DNN performance evaluation is collected. This process is repeated thousands of times until the collected DNN performance distribution converges. Following this process, existing practices [9, 12, 21, 34, 37] generally include up to 10,000 MC runs, which is extremely time-consuming. Other researchers use Bayesian Neural Networks (BNNs) to evaluate the robustness against device variations [7], but the variational inference of BNNs is essentially one form of MC simulation.

Based on these evaluation methods, many works have been proposed in the literature to improve the average performance of CiM DNN accelerators under device variations. They fall into two categories: (1) reducing device variations and (2) enhancing DNN robustness. To reduce device variations, a popular option is write-verify [26]. The approach applies iterative write and read (verify) pulses to make sure that the maximum difference between the weights eventually programmed into the devices and the desired values are bounded by a designer-specified threshold. Write-verify can reduce the weight deviation from the ideal value to less than 3%, thus reducing the average accuracy degradation of deployed DNNs to less than 0.5% [26]. To enhance DNN robustness, a variety of approaches exist. For example, neural architecture search is devised [35, 36] to automatically search through a designated search space for DNN architectures that are more robust. Variation-aware training [1, 9, 11], on the other hand, injects device variation-induced weight perturbations in the training process, so that the trained DNN weights are more robust against similar types of variations. Other approaches include on-chip in-situ training [39] that trains DNNs directly on noisy devices and Bayesian Neural Network (BNN)-based approaches that utilizes the variational training process of BNNs to improve DNN robustness [7].<sup>1</sup>

---

<sup>1</sup>This project is supported in part by NSF under grants CNS-1822099, CCF-1919167 and CCF-2028879.



**Figure 1: Comparison of the worst-case (highest) top-1 error identified by 100K MC runs and our method, based on device variation model introduced in Section 3.1. Showing results on four DNNs: LeNet for MNIST, CovNet for CIFAR-10, ResNet-18 for CIFAR-10, and ResNet-18 for Tiny ImageNet.**

One common assumption of all these evaluation and improvement methods is that they focus on the average performance of a DNN under device variations, which may work well in general-purpose scenarios. However, for safety-critical applications where failure could result in loss of life (*e.g.*, medical devices, aircraft flight control, and nuclear systems), significant property damage, or damage to the environment, only focusing on the average performance is not enough. The worst-case performance, regardless of its likelihood, must also be considered [13, 31]. Yet this is a very challenging problem: Given the extremely high dimension of the device variation space, simply running MC simulations in hope to capture the worst-case corner during training will not work. As shown in Fig. 1, for various DNNs for different datasets, even though the MC simulation has converged with 100K runs, the highest DNN top-1 error rate discovered is still much higher than that identified by our method to be discussed later in this paper, where the worst-case error is close to 100%.

Despite the importance of the problem, very little has been explored in the literature. The only related work comes from the security perspective [30], where a weight projected gradient descent attack (PGD) method is developed to find the weight perturbation that can lead to mis-classification of inputs. However, the goal is to generate a successful weight perturbation attack, but not to identify the worst-case scenario under all possible variations.

To fill the gap, in this work we propose an optimization framework that can efficiently and effectively find the worst-case performance of DNNs in a CiM accelerator with maximum weight variations bounded by write-verify. We show that the problem can be formulated as a constrained optimization with non-differentiable objective, which can be relaxed and solved by gradient-based optimization. We then conduct experiments on different networks and datasets under a practical setting commonly used (*i.e.*, each device represents 2 bits of data with write-verify and yields a maximum weight perturbation magnitude of 0.03). Weight PGD [30], the only method of relevance in the literature, identifies worst-case scenarios where the accuracy is similar to that of random guess, while ours can find ones with close to zero accuracy. We then use our framework to evaluate the effectiveness of existing solutions designed to enhance the average accuracy of DNNs under device variations, and see how they improve the worst-case performance. We study two types of remedies: reducing device variations and

enhancing DNN robustness. Experimental results suggest that they either induce significant overhead or are not quite effective, and further research is needed.

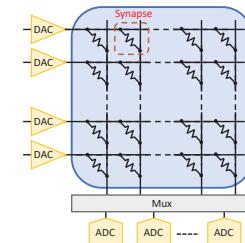
The main contributions of this work are multi-fold:

- This is the first work that formulates the problem of finding worst-case performance in DNN CiM accelerators with device variations for safety-critical applications.
- An efficient gradient-based framework is proposed to solve the non-differentiable optimization problem and find the worst-case scenario.
- Experimental results show that our framework is the only method that can effectively identify the worst-case performance of a DNN.
- We show that even though the maximum weight perturbations are bounded (*e.g.*, by write-verify) to be very small, significant DNN accuracy drop can still occur. Therefore any application of CiM accelerators in the safety-critical settings should use caution.
- We further demonstrate that existing methods to enhance DNN robustness are either too costly or not quite effective in improving worst-case performance. New algorithms in this direction are needed.

The remainder of the paper is organized as follows. In Section 2 we first discuss the background information about CiM DNN accelerators, their robustness issue caused by device variations, and existing methods targeting this issue. We then formulate the problem of finding the worst-case performance of DNN under device variations and propose a framework to solve it in Section 3, along with experimental results to show its efficacy. Extensive studies on the effectiveness of extending existing methods to enhance DNN worst-case performance are carried out in Section 4 and concluding remarks are given in Section 5.

## 2 RELATED WORKS

### 2.1 Crossbar-based Computing Engine



**Figure 2: Illustration of the crossbar architecture. The input is fed horizontally and multiplied by weights stored in the NVM devices at each cross point. The multiplication results are summed up vertically and the sum serves as an output.**

The crossbar array is the key computation engine of CiM DNN accelerators. A crossbar array can perform matrix-vector multiplication in one clock cycle. In a crossbar array, matrix values (*e.g.*, weights in DNNs) are stored at the cross point of each vertical and horizontal line with NVM devices (*e.g.*, RRAMs and FeFETs), and each vector value (*e.g.*, inputs for DNNs) is fed in through horizontal data lines in the form of voltage. The output then flows out

through vertical lines in the form of current. The calculation in the crossbar array is performed in the analog domain according to the Kirchhoff's laws but additional peripheral digital circuits are needed for other key DNN operations (e.g., pooling and non-linear activation), so digital-to-analog and analog-to-digital converters are used between different components especially for DACs to transfer digital input data to voltage levels and ADCs to transfer analog output currents into digital values.

Resistive crossbar arrays suffer from various sources of variations and noises. Two major ones include spatial variations and temporal variations. Spatial variations result from fabrication defects and have both local and global correlations. NVM devices also suffer from temporal variations due to the stochasticity in the device material, which causes fluctuations in conductance when programmed at different times. Temporal variations are typically independent of the device to device and are irrelevant to the value to be programmed [6]. In this work, as a proof of concept, we assume the impact of these non-idealities to be independent on each NVM device. The proposed framework can also be extended to other sources of variations with modification.

## 2.2 Evaluating DNN Robustness Against Device Variations

Most existing researches use a Monte Carlo (MC) simulation-based evaluation process to quantify the robustness of CiM DNN accelerators under the impact of device variations. A device variation model and a circuit model are first extracted from physical measurements. The DNN to be evaluated is then mapped onto the circuit model and the desired value of each NVM device is calculated. In each MC run, for each device, one instance of a non-ideal device is sampled from the variation model, and the actual conductance value of each NVM device is determined. After that, DNN performance (e.g., classification accuracy) can be collected. This process is repeated thousands of times until the collected DNN performance distribution converges. Existing practices [9, 12, 21, 34, 36, 37] generally include close to 10,000 MC runs, which is extremely time-consuming. Empirical results [36, 37] show that 10k MC runs are enough for evaluating the average accuracy of DNNs while no theoretic guarantee is provided.

Several researchers have also looked into the impact of weight perturbations on neural network security [30, 33]. This line of research, dubbed as “Adversarial Weight Perturbation”, tries to link the perturbation in weight to the more thoroughly studied adversarial example issue, where the inputs of DNNs are intentionally perturbed to trigger mis-classifications. One work [33] trains DNNs on adversarial examples to collect adversarial weight perturbation. Most recently, [30] tries to find the adversarial weight perturbation through a modified weight Projected Gradient Descent (PGD) attack. The method can successfully find a small perturbation to reduce the accuracy of DNNs. The work focuses on the success of the attack and does not offer a guarantee of worst-case weight perturbation, as will be demonstrated by our experimental results.

## 2.3 Addressing Device Variations

Various approaches have been proposed to deal with the issue of device variations in CiM DNN accelerators. Here we briefly

review the two most common types: enhancing DNN robustness and reducing device variations.

A common method used to enhance DNN robustness against device variations is variation-aware training [1, 9, 11, 23]. Also known as noise-injection training, the method injects variation to DNN weights in the training process, which can provide a DNN model that is statistically robust against the impact of device variations. In each iteration, in addition to traditional gradient descent, an instance of variation is sampled from a variation distribution and added to the weights in the forward pass. The backpropagation pass is noise free. Once the gradients are collected, this variation is cleared and the variation-free weight is updated according to the previously collected gradients. Other approaches include designing more robust DNN architectures [7, 11, 36, 40] and pruning [2, 12].

To reduce device variations, write-verify [26, 39] is commonly used during the programming process. An NVM device is first programmed to an initial state using a pre-defined pulse pattern. Then the value of the device is read out to verify if its conductance falls within a certain margin from the desired value (*i.e.*, if its value is precise). If not, an additional update pulse is applied, aiming to bring the device conductance closer to the desired one. This process is repeated until the difference between the value programmed into the device and the desired value is acceptable. The process typically requires a few iterations. Most recently, researchers have demonstrated that it is possible to only selectively write-verify a small number of critical devices to maintain the average accuracy of a DNN [34]. There are also various circuit design efforts [10, 18, 27] that try to mitigate the device variations.

## 3 EVALUATING WORST-CASE PERFORMANCE OF CIM DNN ACCELERATORS

A major impact of the device variations is that the conductance of the NVM devices will deviate from the desired value due to the device-to-device variations and cycle-to-cycle variations during the programming process, leading to perturbations in the weight values of a DNN and affecting its accuracy. In Section 3.1, we first model the impact of NVM device variations on weight perturbation, assuming write-verify is used to minimize the variations. Then based on the weight perturbation model, in Section 3.2, we formulate the problem of finding the lowest DNN accuracy under weight perturbation, and devise a framework to solve it. Experimental results are presented in Section 3.3.

### 3.1 Modeling of Weight Perturbation Due to Device Variations

Here we show how we model the impact of device variations on DNN weights. In this paper, we are majorly concerned about the impact of device variations in the programming process, *i.e.*, the conductance value programmed to NVM devices is different from the desired value.

For a weight represented by  $M$  bits, let its desired value  $W_{des}$  be

$$W_{des} = \sum_{j=0}^{M-1} m_j \times 2^j \quad (1)$$

where  $m_j$  is the value of the  $j^{th}$  bit of the desired weight value. Moreover, each NVM device is capable of representing  $K$  bits of data. Thus, each weight value of the DNNs needs  $M/K$  devices to represent<sup>2</sup>. This mapping process can be represented as

$$g_i = \sum_{j=0}^{K-1} m_{i \times K + j} \times 2^j \quad (2)$$

where  $g_i$  is the desired conductance of the  $i^{th}$  device representing a weight. Note that negative weights are mapped in a similar manner. We assume all of the devices use write-verify so that the difference between the actual conductance of each device and its desired value is bounded [6]:

$$gp_i = g_i + n_i, -th \leq n_i \leq th \quad (3)$$

where  $gp_i$  is the actually programmed conductance and  $th$  is the designated write-verify tolerate threshold.

Thus, when a weight is programmed, the actual value  $W_p$  mapped on the devices would be

$$\begin{aligned} W_p &= \sum_{i=0}^{M/K-1} 2^{i \times K} gp_i \\ &= \sum_{i=0}^{M/K-1} 2^{i \times K} g_i + n_i \\ &= W_{des} + \sum_{i=0}^{M/K-1} n_i \times 2^{i \times K} \\ W_{des} - th_g &\leq W_p \leq W_{des} + th_g \end{aligned} \quad (4)$$

where  $th_g = \sum_{i=0}^{M/K-1} (th \times 2^{i \times K})$ . In this paper, we denote  $th_g$  as the *weight perturbation bound*.

In this paper, we set  $K = 2$  as in [11, 34]. Same as the standard practice discussed in Section 2.3, for each weight, we iteratively program the difference between the value on the device and the expected value until it is below 0.1, *i.e.*,  $th = 0.06$  [34], resulting in  $th_g = 0.03$  (unless otherwise specified). These numbers are in line with those reported in [26], which confirms the validity of our model and parameters.

### 3.2 Problem Definition

Now that we have the weight perturbation model, we can define the problem of identifying the worst-case DNN accuracy. Without loss of generality, in this work, we use  $\{f, \mathbf{W}\}$  to represent a neural network, where  $f$  is its neural architecture and  $\mathbf{W}$  is its weights. The forward pass of this neural network is represented by  $f(\mathbf{W}, \mathbf{x})$ , where  $\mathbf{x}$  is the inputs.

From the model in Section 3.1, we can see that the weight perturbation due to device variations is additive and independent. As such, the forward pass of a neural network under the impact of device variation can be expressed as  $f(\mathbf{W} + \Delta\mathbf{W}, \mathbf{x})$ , where  $\Delta\mathbf{W}$  is the weight perturbation caused by device variations. We can define the *perturbed neural network* as  $\{f, \mathbf{W} + \Delta\mathbf{W}\}$ .

With these annotations, we can have the following problem definition: Given a neural network  $\{f, \mathbf{W}\}$  and an evaluation dataset

<sup>2</sup>Without loss of generality, we assume that  $M$  is a multiple of  $K$ .

$D$ , find the perturbation  $\Delta\mathbf{W}$  that the accuracy of perturbed neural network  $\{f, \mathbf{W} + \Delta\mathbf{W}\}$  in dataset  $D$  is the lowest among all possible perturbations inside the weight perturbation bound. In the rest of this paper, this perturbation  $\Delta\mathbf{W}$  is denoted as the *worst-case weight perturbation*; the resultant performance(accuracy) is denoted as the *worst-case performance(accuracy)*; and the corresponding neural network is denoted as the *worst-case neural network*.

Under this definition, we can formalize the problem as:

$$\begin{aligned} \underset{\Delta\mathbf{W}}{\text{minimize}} \quad & | \{f(\mathbf{W} + \Delta\mathbf{W}, \mathbf{x}) == t | (x, t) \in D\} | \\ \text{s.t.} \quad & \mathcal{L}(\Delta\mathbf{W}) \leq th_g \end{aligned} \quad (5)$$

where  $x$  and  $t$  are the input data and classification label in dataset  $D$ , respectively.  $\mathcal{L}(\Delta\mathbf{W})$  is the maximum magnitude of weight perturbation, *i.e.*,  $\max(\text{abs}(\Delta\mathbf{W}))$ ,  $th_g$  is the weight perturbation bound in (4) in Section 3.1, and  $|A|$  denotes the cardinality (size) of a set  $A$ . As  $f$ ,  $\mathbf{W}$  and  $D$  are fixed, the goal is to find the  $\Delta\mathbf{W}$  that minimizes the size of this set of correct classifications, *i.e.*, achieving the worst-case accuracy.

### 3.3 Finding the Worst-Case Performance

The optimization problem defined by (5) is extremely difficult to solve directly due to the non-differentiable objective function. In this section, we put forward a framework to cast it into an alternative form that can be solved by existing optimization algorithms.

To start with, we can slightly relax the objective. Consider a function  $p$  such that  $f(\mathbf{W} + \Delta\mathbf{W}, \mathbf{x}) == t$  if and only if  $p(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) \geq 0$ . In this case, the optimization objective

$$|f(\mathbf{W} + \Delta\mathbf{W}, \mathbf{x}) == t | (x, t) \in D\} | \quad (6)$$

can be relaxed to

$$\sum_{\mathbf{x} \in D} p(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) \quad (7)$$

Intuitively, minimizing (7) can help to minimize (6), and these two optimization problems become strictly equivalent if all data in  $D$  is mis-classified in the presence of  $\Delta\mathbf{W}$ .

There are various choices of  $p(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\})$  that can meet the requirement. We show some of the representative ones below.

$$\begin{aligned} \mathbf{O} &= f(\mathbf{W} + \Delta\mathbf{W}, \mathbf{x}) \\ \mathbf{Z} &= \text{Softmax}(f(\mathbf{W} + \Delta\mathbf{W}, \mathbf{x})) \\ p_1(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) &= -\text{loss}(\mathbf{O}, t) + 1 \\ p_2(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) &= \max_{i \neq t} \{\max(O_i) - O_t, 0\} \\ p_3(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) &= \text{softplus}(\max_{i \neq t} (O_i) - O_t) - \log(2) \\ p_4(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) &= \max\{0.5 - O_t, 0\} \\ p_5(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) &= -\log(2 \cdot O_t - 2) \\ p_6(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) &= \max_{i \neq t} \{\max(Z_i) - Z_t, 0\} \\ p_7(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) &= \text{softplus}(\max_{i \neq t} (Z_i) - Z_t) - \log(2) \end{aligned} \quad (8)$$

where  $x$  and  $t$  are the input data and classification label, respectively.  $\text{softplus}(x) = \log(1 + \exp(x))$  and  $\text{loss}(\mathbf{O}, t)$  is cross entropy loss.

According to the empirical results, in this paper we choose

$$\begin{aligned} \mathbf{O} &= f(\mathbf{W} + \Delta\mathbf{W}, \mathbf{x}) \\ p(\mathbf{x}, \{f, \mathbf{W} + \Delta\mathbf{W}\}) &= \max_{i \neq t} \{\max(O_i) - O_t, 0\} \end{aligned} \quad (9)$$

We now have the relaxed optimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{x \in D} p(x, \{f, \mathbf{W} + \Delta \mathbf{W}\}) \\ \text{s.t.} \quad & \mathcal{L}(\Delta \mathbf{W}) \leq th_g \end{aligned} \quad (10)$$

For this relaxed problem, we can utilize Lagrange multiplier to provide an alternative formulation

$$\text{minimize} \left( c \cdot \sum_{x \in D} p(x, \{f, \mathbf{W} + \Delta \mathbf{W}\}) + (\mathcal{L}(\Delta \mathbf{W}) - th_g) \right) \quad (11)$$

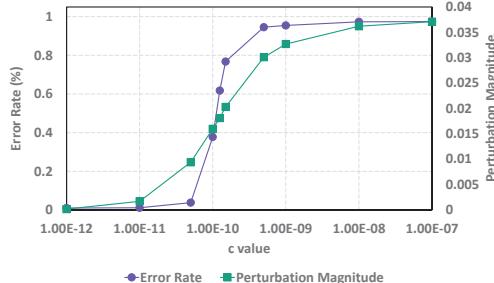
where  $c > 0$  is a suitably chosen constant, if optimal solution exists. This objective is equivalent to the relaxed problem, in the sense that there exists  $c > 0$  such that the optimal solution to the latter matches the optimal solution to the former.

Thus, we use the optimization objective (11) as the relaxed alternative objective of the defined objective (5). Because the objective (11) is differentiable w.r.t.  $\Delta \mathbf{W}$ , we use gradient descent as the optimization algorithm to solve this problem.

#### The choice of constant $c$ .

Qualitatively speaking, observing objective (11) with larger  $c$  values means more focus on lower accuracy and less focus on  $\mathcal{L}(\Delta \mathbf{W})$ , which would result in lower final accuracy and greater  $\mathcal{L}(\Delta \mathbf{W})$ . The empirical results shown in Fig. 3 also prove this observation, where we plot how the worst-case error rate and  $\mathcal{L}(\Delta \mathbf{W})$  varies with the choice of  $c$  using LeNet for MNIST.

Because the empirical result show that  $\mathcal{L}(\Delta \mathbf{W})$  is monotonic w.r.t.  $c$ , to find the  $c$  value that leads to the lowest performance under the weight perturbation bound  $th_g$ , we use binary search to find the largest  $c$  value that ensures  $\mathcal{L}(\Delta \mathbf{W}) \leq th_g$ . The corresponding accuracy obtained with this  $c$  is then the worst-case performance of this DNN model under weight perturbations bounded by  $th_g$ .



**Figure 3: Choice of the constant  $c$ .** We plot two different relations: (1) Worst-case error rate (left) w.r.t.  $c$  value and (2) perturbation magnitude  $\mathcal{L}(\Delta \mathbf{W})$  (right) w.r.t.  $c$  value. The DNN model used is LeNet for MNIST.

Finally, since the optimization problem stated in (11) can be solved via gradient descent, the time and memory complexity of our algorithm is comparable to that needed to train the DNN.

## 3.4 Experimental Evaluation

In this section, we use experiments to demonstrate the effectiveness of the proposed method in finding the worst-case performance of different DNN models. Six different DNN models for four different datasets are used: (1) LeNet [17] for MNIST [5], (2) ConvNet [23] for CIFAR-10 [15], (3) ResNet-18 [8] for CIFAR-10, (4) ResNet-18 for Tiny ImageNet [16], (5) ResNet-18 for ImageNet [4] and (6) VGG-16 [28] for ImageNet. LeNet and ConvNet are quantized to 4 bits

**Table 1: Hyper-parameter setups to perform the proposed method on different models for different datasets. The  $c$  value in (11) identified by binary search, the learning rate (lr) and the number of iterations used for gradient descent are specified.**

Dataset	Model	$c$	lr	# of runs
MNIST	LeNet	1E-3	1E-5	500
CIFAR-10	ConvNet	1E-5	1E-5	100
CIFAR-10	ResNet-18	1E-9	1E-4	20
Tiny ImgNet	ResNet-18	1E-10	1E-4	20
ImageNet	ResNet-18	1E-3	1E-3	10
ImageNet	VGG-16	1E-3	1E-3	10

while ResNet-18 and VGG-16 are quantized to 8 bits. As discussed in Section 3.1, we use  $th_g = 0.03$  as the weight perturbation bound, i.e., each weight is perturbed by at most  $\pm 0.03$ .

As there is no existing work on identifying the worst-case performance of a DNN under device variations to compare with other than the naive MC simulations, we slightly modify the weight PGD attack method [30], which tries to find the smallest weight perturbation that can lead to a successful attack, as an additional baseline. Experiments are conducted on Titan-XP GPUs in the PyTorch framework. For MC simulation baseline, 100,000 runs are used. We use Adam [14] as the gradient descent optimizer. The detailed setup for the proposed method is shown in Table 1.

**3.4.1 Worst-case DNN Accuracy Obtained by Different Methods.** As shown in Table 2, compared with weight PGD attack and MC simulations, the proposed framework is more effective in finding the worst-case performance. It identifies worst-case weight perturbations that can lead to below 10% accuracy for LeNet and ConvNet, and almost 0% accuracy for ResNet-18 and VGG-16. On the other hand, the weight PGD attack can only find perturbations that lead to DNN accuracy close to random guessing (i.e., 1/N for N classes, which is 10% for CIFAR-10, 0.5% for Tiny ImageNet, and 0.1% for ImageNet). MC simulations perform the worst. With 100,000 runs it fails to find any perturbation that can result in accuracy drop similar to those of the other two methods. This is quite expected given the high dimensional exploration space spanned by the large number of weights. Our framework takes slightly longer time to run than the weight PGD attack method, mainly due to the number of epochs the gradient descent takes to converge. Yet both methods are much faster than the MC simulations.

The results from the table suggest that DNNs are extremely vulnerable to device variations, even though write-verify is used and the maximum weight perturbation is only 0.03. Considering the fact that even converged 100,000 MC simulations cannot get close to the actual worst-case accuracy, For safety-critical applications, it may be necessary to screen each programmed CiM accelerator and test its accuracy to avoid disastrous consequences. Random sampling based quality control may not be an option.

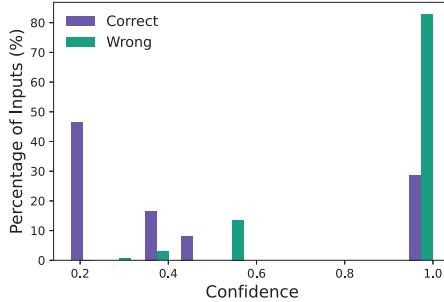
In addition, comparing the results obtained by our framework on ConvNet and ResNet-18 for CIFAR-10 (as well as ResNet-18 and VGG-16 for ImageNet) we can see that deeper networks are more susceptible to weight perturbations. This is expected as more perturbation can be accumulated in the forward propagation.

**Table 2: Comparison between MC simulation (MC), weight PGD attack (PGD) and the proposed framework in obtaining the worst-case accuracy of various DNN models for different dataset using weight perturbation bound  $th_g = 0.03$ . The accuracy of the original model without perturbation (Ori. Acc) is also provided. The proposed method finds perturbations that lead to much lower accuracy than those found by other methods, using slightly longer time than the weight PGD attack method but much shorter time than the MC simulation.**

Dataset	Model	Ori. Acc.	Worst-case Accuracy (%)			Time (Minutes)		
			MC	PGD	Proposed	MC	PGD	Proposed
MNIST	LeNet	99.12	97.34	13.44	<b>7.35</b>	900	3.3	5.5
CIFAR-10	ConvNet	85.31	60.12	10.00	<b>4.27</b>	2700	4.2	6.0
CIFAR-10	ResNet-18	95.14	88.77	10.00	<b>0.00</b>	5400	13.3	20.0
Tiny ImageNet	ResNet-18	65.23	25.33	0.50	<b>0.00</b>	14400	40.0	60.0
ImageNet	ResNet-18	69.75	43.98	0.10	<b>0.00</b>	231000	1980	2880
ImageNet	VGG-16	71.59	66.43	0.10	<b>0.06</b>	313800	2530	3820

Finally, the experimental results also show that quantization in both weights and activations is not an effective method to improve worst-case DNN performance, because all the models in these experiments are quantized as explained in the experimental setup.

### 3.4.2 Analysis of Classification Results.



**Figure 4: Distribution of classification confidence among correct/wrong cases from the worst-case LeNet for MNIST. The model is more confident in the wrong cases than in the correct ones.**

We now take a closer look at the classification results of the worst-case LeNet for MNIST identified by our framework. We first examine the classification confidence, the distribution of which is shown in Fig. 4. Same as the common practice, the classification confidence of a DNN on an input is calculated by a Softmax function on its output vector. The element having the highest confidence is considered the classification result. Contrary to our intuition, from the figure we can see that the worst-case LeNet is highly confident in the inputs it mis-classifies, with an average confidence of 0.90 on all the inputs that are classified wrong. On the other hand, the DNN model is not confident in the inputs it classifies correctly, having an average confidence of only 0.47 on these inputs. This is significantly different from the original LeNet without perturbation, whose confidence is always close to 1.

We also observe how the classification results are distributed among different classes, which are reported in Table. 3. From the table we can see that most of the errors are due to images being wrongly classified to the same class (class 1), while many of the images that truly belong to this class are being classified to other classes (class 2 and class 3).

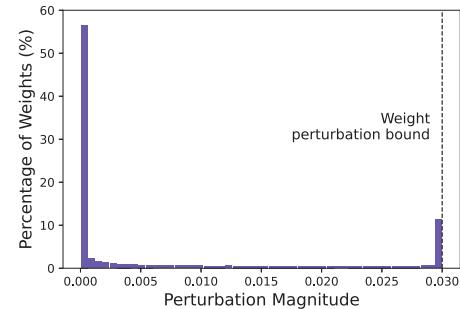
We hope that these observations can potentially shed light on the development of new algorithms to boost the worst-case performance of DNNs in the future.

**Table 3: Normalized classification results of the worst-case LeNet for MNIST. The number in row  $i$  and column  $j$  indicates how many cases with class  $i$  as ground truth are being classified as  $j$ , normalized over the total number of cases in class  $i$ . Most inputs are mis-classified to one class (class 1).**

Ground Truth	Classification Result									
	0	1	2	3	4	5	6	7	8	9
0	0.0	<b>1.0</b>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.4	<b>0.5</b>	<b>0.1</b>	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	<b>1.0</b>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	<b>1.0</b>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	<b>0.9</b>	0.0	0.0	0.0	0.0	0.0	<b>0.1</b>	0.0	0.0
5	0.0	<b>0.9</b>	0.0	<b>0.1</b>	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	<b>1.0</b>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	<b>0.6</b>	0.0	<b>0.4</b>	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	<b>0.8</b>	0.0	<b>0.2</b>	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	<b>0.9</b>	0.0	<b>0.1</b>	0.0	0.0	0.0	0.0	0.0	0.0

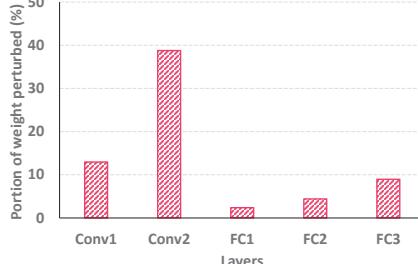
### 3.4.3 Distribution of Worst-Case Weight Perturbation.

Here we show how the perturbation is distributed among the weights in the worst-case LeNet for MNIST. As can be seen in Fig. 5, most of the weights are either not perturbed or perturbed to the maximum magnitude (*i.e.*,  $th_g = 0.03$ ).



**Figure 5: The distribution of the weight perturbation magnitude in the worst-case LeNet for MNIST. Most weights are either not perturbed or perturbed to  $th_g$ .**

We then show the number of weights that are perturbed in each layer in Fig. 6. We can observe that the weights in convolutional layers and the final FC layer are more likely to be perturbed. This is probably due to the fact that they in general have more impact on the accuracy of a DNN.



**Figure 6: The percentage of weights being perturbed in each layer of the worst-case LeNet for MNIST. Weights in convolutional layers and the last FC layer are more likely to be perturbed.**

## 4 ENHANCING WORST-CASE PERFORMANCE OF CIM DNN ACCELERATORS

Several works exist in the literature to improve the average performance of a DNN under device variations. In this section, we try to extend them to improve the worst-case performance of a DNN, and evaluate their effectiveness. Specifically, we will include two types of methods: (1) confining the device variations and (2) training DNN models that are more robust against device variations. As discussed in Section 2, one of the most popular practices of the former is write-verify and the latter includes variation-aware training.

In addition to these, we also modify adversarial training [30], a method commonly used to combat adversarial input, to address weight perturbation caused by device variations. The algorithm is summarized in Alg. 1. Similar to how adversarial training handles input perturbations, in DNN training process we inject worst-case perturbations to the weights of a DNN, in hope that they perform better under the impact of device variations. Specifically, in each iteration of training, we first conduct the proposed method to find the perturbations of the current weights that can lead to worst-case accuracy of the model. We then add them to the weights and collect the gradient  $G$ .

If not specified explicitly, all accuracy results shown in this section are collected by training one DNN architecture using the same specification but with **three** different initializations. The accuracy (error rate) number is shown in percentage and presented in [average  $\pm$  standard deviation] for these three runs.

### 4.1 Stronger Write-Verify

As shown in Table 2, using the standard write-verify setting in the literature to confine the maximum weight perturbation to 0.03 ( $th_g = 0.03$  in (4)) cannot significantly improve the worst-case performance of DNN models. If we set a smaller  $th_g$  in write-verify, the write time would become longer but can potentially help to boost the worst-case performance.

To see the relationship between  $th_g$  and worst-case DNN accuracy, we use three models, *i.e.*, LeNet for MNIST, ConvNet for CIFAR-10, and ResNet-18 for CIFAR-10, and plot the results as

---

### Algorithm 1 Adversarial Training ( $f, D, V, Ep, l, \eta, C$ )

---

```

1: // INPUT: A DNN architecture  $f$ , training dataset D, validation
   dataset V, the total number of training epochs  $Ep$ , loss function
    $l$ , learning rate  $\eta$ ;
2: Initialize weight  $W$  for  $f$ ;
3: Initialize  $acc_F = 0$ ,  $W_F = W$ ;
4: for ( $i = 0$ ;  $i < Ep$ ;  $i++$ ) do
5:   for mini-batches B in D do
6:     Divide B into input I and label L;
7:     Find weight perturbations N that lead to worst-case accuracy
       using the framework discussed in Section 3.3;
8:      $O = f(W + N, I)$ ;
9:      $loss = l(O, L)$ ;
10:     $G = \frac{\partial loss}{\partial W}$ ;
11:     $W = W - \eta \times G$ 
12:  end for
13:  Evaluate W on V and get  $acc$ ;
14:  if  $acc > acc_F$  then
15:     $acc_F = acc$ ;
16:     $W_F = W$ 
17:  end if
18: end for
19: Return  $W_F$ 

```

---

**Table 4: Worst-case accuracy (%) of various DNN models from regular training (Regular), variation-aware training (VA) and adversarial training (ADV). Write-verify with weight perturbation bound  $th_g = 0.03$ . Compared with regular training, adversarial training is effective in LeNet for MNIST, but both methods are not effective in other more complex models.**

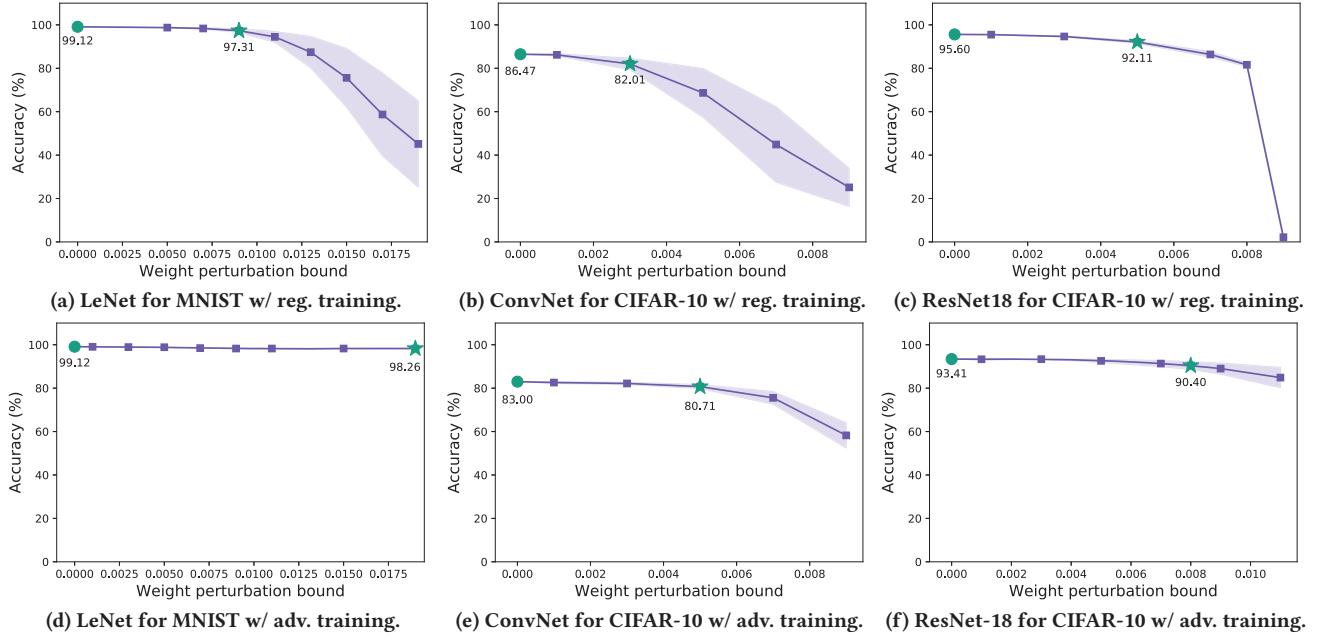
Dataset	Model	Regular	VA	ADV
MNIST	LeNet	$7.35 \pm 0.37$	$18.58 \pm 0.80$	$98.26 \pm 0.05$
CIFAR10	ConvNet	$4.27 \pm 0.33$	$63.71 \pm 3.76$	$67.09 \pm 0.85$
CIFAR10	ResNet18	$0.00 \pm 0.00$	$32.84 \pm 17.20$	$34.84 \pm 13.20$
Tiny IN	ResNet18	$0.00 \pm 0.00$	$3.57 \pm 0.48$	$7.41 \pm 0.10$

shown in Fig. 7(a)-(c), where we also include the model accuracy without any device variation ( $th_g = 0$ ). From the figures we can see that a lower  $th_g$  can indeed increase the worst-case accuracy. Yet to ensure the models have acceptable accuracy in worst-case (e.g., no more than 5% accuracy drop from DNNs without the impact of device variations and marked with star in each figure), we need to set  $th_g = 0.009$  for LeNet,  $th_g = 0.003$  for ConvNet and  $th_g = 0.005$  for ResNet-18, which would take extremely long write time. The experimental result of ResNet-18 for Tiny ImageNet is not shown here because its worst-case accuracy is lower than 20% even when  $th_g = 0.001$  and further reducing  $th_g$  is not practical.

### 4.2 Variation-aware and Adversarial Training

Here we study the effectiveness of variation-aware training and adversarial training on four models: LeNet for MNIST, ConvNet for CIFAR-10, ResNet-18 for CIFAR-10, and ResNet-18 for Tiny ImageNet. We assume that standard write-verify with weight perturbation bound  $th_g = 0.03$  is used for all the models.

As shown in Table 4, both variation-aware training and adversarial training can offer some improvements in most cases compared



**Figure 7: Effectiveness of write-verify with regular training (a)-(c), and with adversarial training (d)-(f). Figures represent the relationship between weight perturbation bound in write-verify  $th_g$  (X-axis) and the worst-case DNN accuracy (Y-axis) in different models: (a)(e) LeNet for MNIST, (b)(d) ConvNet for CIFAR-10, and (c)(f) ResNet-18 for CIFAR-10. For each data point, three experiments of the same setting but different random initialization are conducted. The solid lines show the averaged results over the three experiments and the shadows represent the standard deviations. In each figure, the circle marks the model without perturbation ( $th_g = 0$ ) and the star marks the model with highest  $th_g$  and no more than 5% accuracy degradation.**

with the regular training. Adversarial training is slightly more effective than variation-aware training. However, compared with the accuracy that can be obtained by these networks without device variations (third column in Table 2), the accuracy drop is still significant in the worst case. The only exception is the case of LeNet for MNIST, where adversarial training can almost fully recover the accuracy loss even in the worst case, thanks to its simplicity. In addition, we can observe that as the network gets deeper, the worst-case accuracy improvement brought by these two training methods starts to diminish (e.g. 7.41% for ResNet-18 for Tiny ImageNet).

### 4.3 Combining Adversarial Training with Write-Verify

Finally, using the same three models and datasets, we show whether the models trained by the adversarial training method can reduce the requirement on write-verify to achieve the same worst-case accuracy. The results are shown in Fig. 7(d)-(f). Comparing with the results of the models from regular training in Fig. 7(a)-(c), with adversarial training, the weight perturbation bound  $th_g$  needed to achieve the same accuracy increases. As discussed in Section 4.2, with adversarial training, the worst-case accuracy of LeNet for MNIST using the standard write-verify ( $th_g = 0.03$ ) is already very close to that of the original model without device variations. Thus, Fig. 7 (d) is almost flat. For the other two models, to ensure no more than 5% worst-case accuracy degradation from the original model without device variations, we now need  $th_g = 0.005$  for ConvNet for CIFAR-10, and  $th_g = 0.008$  for ResNet-18 for CIFAR-10, as marked by the star in each figure. Comparing with the weight perturbation

bound needed to attain the same worst-case accuracy in Fig 7(b)-(c), we can see that using adversarial training instead of regular training can increase it by around 1.7×, indicating faster programming process. However, these bounds are still much smaller than the commonly used 0.03 [11, 26] and take considerably more programming time. Therefore, more effective methods to address the worst-case accuracy are still needed, and calls for future research.

## 5 CONCLUSIONS

In this work, contrary to the existing methods that evaluate the average performance of DNNs under device variations in CiM accelerators, we proposed an efficient framework to examine their worst-case performance, which is important for safety-critical applications. With the proposed framework, we show that even with bounded small weight perturbations after write-verify, the accuracy of a well-trained DNN can drop drastically to almost zero. As such, we should use caution when applying CiM accelerators to safety-critical applications. For example, we may need to screen the accuracy of each chip rather than random sampling in quality control. We further show that the existing methods used to enhance average DNN performance in CiM accelerators are either too costly (for stronger write-verify) or ineffective (for training-based methods) when extended to enhance the worst-case performance. Further research from the community is needed to address this problem.

## REFERENCES

[1] Chih-Cheng Chang, Ming-Hung Wu, Jia-Wei Lin, Chun-Hsien Li, Vivek Parmar, Heng-Yuan Lee, Jeng-Hua Wei, Shyh-Shyuan Sheu, Manan Suri, Tian-Sheuan Chang, et al. 2019. NV-BNN: An accurate deep convolutional neural network based on binary STT-MRAM for adaptive AI edge. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[2] Ching-Yuan Chen and Krishnendu Chakrabarty. 2021. Pruning of Deep Neural Networks for Fault-Tolerant Memristor-based Accelerators. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 889–894.

[3] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH computer architecture news* 44, 3 (2016), 367–379.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[5] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[6] Ben Feinberg, Shibo Wang, and Engin Ipek. 2018. Making memristive neural network accelerators reliable. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 52–65.

[7] Di Gao, Qingrong Huang, Grace Li Zhang, Xunzhao Yin, Bing Li, Ulf Schlichtmann, and Cheng Zhuo. 2021. Bayesian inference based robust computing on memristor crossbar. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 121–126.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Zhezhi He, Jie Lin, Rickard Ewetz, Jiann-Shiun Yuan, and Deliang Fan. 2019. Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[10] Soyoun Jeong, Jaerok Kim, Minhyeok Jeong, and Yoommyung Lee. 2022. Variation-Tolerant and Low R-Ratio Compute-in-Memory ReRAM Macro With Capacitive Ternary MAC Operation. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2022).

[11] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi. 2020. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Trans. Comput.* 70, 4 (2020), 595–605.

[12] Song Jin, Songwei Pei, and Yu Wang. 2020. On improving fault tolerance of memristor crossbar based neural network designs by target sparsifying. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 91–96.

[13] Brendan Juba and Hengxuan Li. 2020. More Accurate Learning of k-DNF Reference Classes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4385–4393.

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[16] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[18] Wantong Li, Xiaoyu Sun, Hongwu Jiang, Shanshi Huang, and Shimeng Yu. 2021. A 40nm RRAM Compute-in-Memory Macro Featuring On-Chip Write-Verify and Offset-Cancelling ADC References. In *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 79–82.

[19] Zhdining Liang, Hanrui Wang, Jinglei Cheng, Yongshan Ding, Hang Ren, Xuehai Qian, Song Han, Weiwen Jiang, and Yiyu Shi. 2022. Variational quantum pulse learning. *arXiv preprint arXiv:2203.17267* (2022).

[20] Zhdining Liang, Zhepeng Wang, Junhuan Yang, Lei Yang, Yiyu Shi, and Weiwen Jiang. 2021. Can noise on qubits be learned in quantum neural network? a case study on quantumflow. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.

[21] Tao Liu, Wujie Wen, Lei Jiang, Yanzhi Wang, Chengmo Yang, and Gang Quan. 2019. A fault-tolerant neural network architecture. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[22] Bingqian Lu, Zheyu Yan, Yiyu Shi, and Shaolei Ren. 2022. A Semi-Decoupled Approach to Fast and Optimal Hardware-Software Co-Design of Neural Accelerators. *arXiv preprint arXiv:2203.13921* (2022).

[23] Xiaochen Peng, Shanshi Huang, Yandong Luo, Xiaoyu Sun, and Shimeng Yu. 2019. DNN+ NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. In *2019 IEEE international electron devices meeting (IEDM)*. IEEE, 32–5.

[24] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[25] Yi Sheng, Junhuan Yang, Yawen Wu, Kevin Mao, Yiyu Shi, Jingtong Hu, Weiwen Jiang, and Lei Yang. 2022. The Larger The Fairer? Small Neural Networks Can Achieve Fairness for Edge Devices. (2022).

[26] Wonbo Shim, Jae-sun Seo, and Shimeng Yu. 2020. Two-step write-verify scheme and impact of the read noise in multilevel RRAM-based inference engine. *Semiconductor Science and Technology* 35, 11 (2020), 115026.

[27] Hyein Shin, Myeonggu Kang, and Lee-Sup Kim. 2021. Fault-free: A Fault-resilient Deep Neural Network Accelerator based on Realistic ReRAM Devices. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1039–1044.

[28] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[29] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[30] Yu-Lin Tsai, Chia-Yi Hsu, Chia-Mu Yu, and Pin-Yu Chen. 2021. Formalizing Generalization and Adversarial Robustness of Neural Networks to Weight Perturbations. *Advances in Neural Information Processing Systems* 34 (2021).

[31] Zhilu Wang, Chao Huang, and Qi Zhu. 2022. Efficient Global Robustness Certification of Neural Networks via Interleaving Twin-Network Encoding. *arXiv preprint arXiv:2203.14141* (2022).

[32] Zhepeng Wang, Zhiding Liang, Shanglin Zhou, Caiwen Ding, Yiyu Shi, and Weiwen Jiang. 2021. Exploration of quantum neural architecture by mixing quantum neuron designs. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.

[33] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. 2020. Adversarial weight perturbation helps robust generalization. *Advances in Neural Information Processing Systems* 33 (2020), 2958–2969.

[34] Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi. 2022. SWIM: Selective Write-Verify for Computing-in-Memory Neural Accelerators. In *2022 59th ACM/IEEE Design Automation Conference (DAC)*. IEEE.

[35] Zheyu Yan, Weiwen Jiang, Xiaobo Sharon Hu, and Yiyu Shi. 2022. RADARS: Memory Efficient Reinforcement Learning Aided Differentiable Neural Architecture Search. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 128–133.

[36] Zheyu Yan, Da-Cheng Juan, Xiaobo Sharon Hu, and Yiyu Shi. 2021. Uncertainty Modeling of Emerging Device based Computing-in-Memory Neural Accelerators with Application to Neural Architecture Search. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 859–864.

[37] Zheyu Yan, Yiyu Shi, Wang Liao, Masanori Hashimoto, Xichuan Zhou, and Cheng Zhuo. 2020. When single event upset meets deep neural networks: Observations, explorations, and remedies. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 163–168.

[38] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[39] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. 2020. Fully hardware-implemented memristor convolutional neural network. *Nature* 577, 7792 (2020), 641–646.

[40] Nanyang Ye, Jingbiao Mei, Zhicheng Fang, Yuwen Zhang, Ziqing Zhang, Huaying Wu, and Xiaoyao Liang. 2021. BayesFT: Bayesian Optimization for Fault Tolerant Neural Network Architecture. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 487–492.