

# Tolerating Defects in Low-Power Neural Network Accelerators Via Retraining-Free Weight Approximation

Fateme S. Hosseini, Fanruo Meng, and Chengmo Yang,  
University of Delaware, USA  
Wujie Wen, Lehigh University, USA  
Rosario Cammarota, Intel, USA

Authors' addresses: F. S. Hosseini, F. Meng, and C. Yang, Department of Electrical and Computer Engineering, University of Delaware, Newark, DE 19716 USA; emails: {fateme, mengfanr, chengmo}@udel.edu; W. Wen, Department of Electrical and Computer Engineering, Lehigh University, Bethlehem, PA 18015, USA; email: wuw219@lehigh.edu; R. Cammarota, Intel, San Jose, CA 95134; email: rosario.cammarota@intel.com.

© 2021 Association for Computing Machinery.

This work is supported by Semiconductor Research Corporation grant #2964.001, National Science Foundation grants #1909854 and #2011236.

Hardware accelerators are essential to the accommodation of ever-increasing Deep Neural Network (DNN) workloads on the resource-constrained embedded devices. While accelerators facilitate fast and energy-efficient DNN operations, their accuracy is threatened by faults in their on-chip and off-chip memories, where millions of DNN weights are held. The use of emerging Non-Volatile Memories (NVM) further exposes DNN accelerators to a non-negligible rate of permanent defects due to immature fabrication, limited endurance, and aging. To tolerate defects in NVM-based DNN accelerators, previous work either requires extra redundancy in hardware or performs defect-aware retraining, imposing significant overhead. In comparison, this paper proposes a set of algorithms that exploit the flexibility in setting the fault-free bits in weight memory to effectively approximate weight values, so as to mitigate defect-induced accuracy drop. These algorithms can be applied as a one-step solution when loading the weights to embedded devices. They only require trivial hardware support and impose negligible run-time overhead. Experiments on popular DNN models show that the proposed techniques successfully boost inference accuracy even in the face of elevated defect rates in the weight memory.

**CCS Concepts:** Computer systems organization → Reliability; Neural networks; *Embedded software*; Hardware → Error detection and error correction;

**Additional Key Words and Phrases:** Neural network accelerator, defect tolerance, reliability, memory faults, approximation

## 1 INTRODUCTION

Deep Neural Networks (DNNs) have gained increasing popularity in many real-world applications spanning from image recognition and natural language processing, to life sciences, genomics, self-driving cars, and big data analytics. While the opportunities seem boundless, the harsh reality is that DNN models not only require the storage for millions of weights but also drastically escalate the system’s power/energy consumption. To accommodate more DNN workloads on resource-constrained embedded devices, a lot of research attention has been paid to develop energy-efficient hardware accelerators, including both traditional ASIC and FPGA-based accelerators [10, 16, 27, 44] as well as emerging processing-in-memory (PIM) accelerators [14, 24, 40].

The performance and energy consumption of a DNN accelerator are largely determined by its underlying memory technology and architecture. As traditional SRAM and DRAM technologies are power-hungry, emerging Non-Volatile Memory (NVM) technologies such as Phase Change Memory (PCM) [2], Resistive RAM (RRAM) [46], and Spin-Transfer Torque RAM (STT-RAM) [31] have been proposed for their usage in battery-constrained embedded devices. The *in-situ* computing ability of NVMs also makes them ideal for conducting Multiplication-and-Accumulation (MAC) – the fundamental computation in DNNs. Despite these benefits, NVMs are known for their slow write speed, high write energy, and limited write endurance [8, 15, 18, 32, 49]. To benefit from different memory technologies while overcoming their limitations, hybrid memory architectures have been proposed [18, 23, 37]. Such designs are well-suited for DNN accelerators, as SRAM/DRAM can be used to buffer the frequently updated intermediate results to offset NVM write performance, energy and endurance limitations, while NVMs can be used to hold weight values that are meant to be reused and hardly updated when classifying different images [9, 29, 33, 42].

From the reliability point of view, traditional and emerging memory technologies are subject to different challenges as well. Traditional SRAM and DRAM, given their nanoscale feature size and reduced noise margin, are exposed to *transient faults* caused by alpha-particle strikes, cosmic rays, or radiation from radioactive atoms [30], while *permanent faults* are not a major concern given their relatively mature manufacturing process (~12 failures in 1 billion hours [6]). Emerging NVMs, on the other hand, are less vulnerable to transient faults but suffer from a more significant rate of permanent faults (also called *defects*<sup>1</sup>) due to immature fabrication, imprecise programming, limited write endurance, and aging [1, 3], e.g., under the impact of process variation, the permanent fault rate of PCM cells can go up to 1% due to the limited endurance [28]. As the accelerator ages, defects can accumulate in the memory, resulting in a dramatic accuracy drop that turns the DNN output to random guesses [48].

While transient faults in SRAM/DRAM can be detected and/or corrected via error correction codes (ECC), ECC is not desirable for tolerating permanent faults, where the state of a memory cell is stuck at either 0 or 1. The constant bit-checking and correction process enforced by the unrecoverable nature of defects would result in significant performance and energy cost [36] – even for DRAM, ECC is reported to cause 2.5X slow down in the average execution time [12]. What’s worse, ECC only offers limited error correction capability. If a memory line has more than one stuck-at-faults, standard ECC will incorrectly modify a non-faulty bit. For these reasons, ECCs are not preferable for tolerating permanent faults in NVMs, and other techniques such as Error-correcting-pointers (ECPs) [38], defect-aware mapping, pruning, or retraining have been proposed. However, none of them is a proper fit for resource-constrained devices as they either rely on performing extremely expensive, time-consuming, and unscalable retraining process [4, 25, 26, 48, 52] or require an extra level of redundancy in the memory or complex routing operations [4, 19, 21, 25, 38, 47].

---

<sup>1</sup>In this paper, the terms of faults and defects both refer to permanent hardware faults and are used interchangeably.

In comparison, the goal of this work is to develop a light-weight and retraining-free solution to mitigate the accuracy drop in DNN models induced by defects in NVMs. Our work is driven by two major characteristics of resource-constrained DNN accelerators: limited memory space which demands full access to all the available memory cells without eliminating defective ones, and limited battery lifetime which favors accessing contiguous memory space where DNN weights are not selectively remapped to different locations. To meet these requirements, we propose to *utilize* defective memory cells and *mitigate* their adverse impact via value approximation. Specifically, we develop three algorithms to reduce the deviation of DNN weights from their expected values via careful manipulation of the fault-free bits. This deviation reduction process is performed before loading the weights to the hardware, i.e., at deployment-time, thus minimizing the associated hardware and runtime overhead.

The rest of this paper is organized as follows: Section 2 reviews the related work on defect tolerance in neural network accelerators and highlights the main differences with our work. Section 3 provides the motivation behind this work and describes technical details of the proposed algorithms. Section 4 presents the experimental setup and results, while Section 5 concludes the paper.

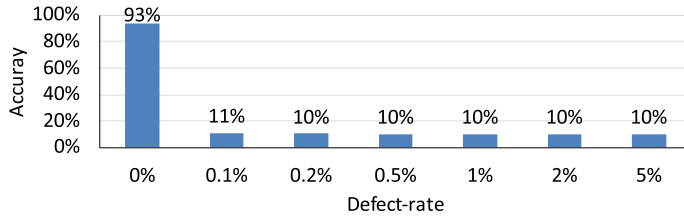
## 2 PRELIMINARIES

### 2.1 Impact of Faults on DNN Inference Accuracy

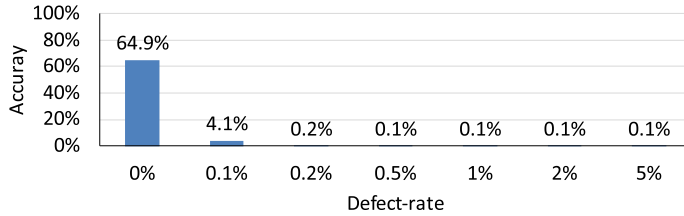
As mentioned before, a DNN accelerator with hybrid memory architecture is mainly subject to two types of faults, *transient faults* in SRAM/DRAM which affect intermediate computation results, and as well as *permanent faults* in the emerging NVM which may corrupt weight values. This work focuses on the second type for two major reasons. First, transient faults in the intermediate results only affect the inference of one input, whereas the impact of faults in weight values is persistent to all inputs. Second, unlike transient faults that can be detected and/or corrected via error correction codes (ECC), permanent faults are much difficult to tolerate due to their unrecoverable nature.

As a result of their immature manufacturing, imprecise programming, limited write endurance, and aging [1, 3, 28], the rates of permanent defects in emerging non-volatile memory technologies are expected to be high. In [28], it is reported that the rate of stuck-at-faults in PCM can quickly reach 1% due to the limited endurance. The defect rate in other NVM technologies can be even higher, e.g., RRAM devices are reported to show up to 10.79% permanent faults (9.04% stuck-at-1s and 1.75% stuck-at-0s) [1, 3]. While traditionally neural networks are expected to show graceful degradation in their accuracy in the presence of noisy inputs or small variations/errors in the underlying hardware [39, 43], this expectation no longer holds in the face of the elevated fault rates in emerging NVMs. In fact, recent work shows that DNN accuracy can be significantly affected by the accumulation of hardware faults [13, 34, 48, 50]. As a concrete example, Figure 1 illustrates the impact of faults in network weights on the accuracy of two popular DNNs trained and tested on well-known datasets: VGG-16 for CIFAR-10 dataset (with 10 output classes) and Resnet-18 for Imagenet dataset (with 1000 output classes) both under 16-bit quantization level.<sup>2</sup> The models are evaluated under fault rates of 0 to 5%, where x% fault-rate means that each memory cell has x% probability to be faulty. Each fault has roughly 50% chance to alter the corresponding quantized weight value (with the other 50% chance of being silent). More details on the experiment setup is described in Section 4.1.1. As shown, VGG-16’s accuracy drops from ~93% in the fault-free case to ~11% under 0.1% fault rate, at which the model becomes practically useless. The accuracy drops to ~10% at higher fault rates which is equivalent to the random guess accuracy for Cifar-10 (and thus

<sup>2</sup>Quantization-level is the number of bits used to represent one weight value in integer/fixed-point format.



(a) 16-bit VGG-16



(b) 16-bit Resnet-18

Fig. 1. Accuracy vs fault-rate in (a) VGG-16 tested on CIFAR-10 dataset, and (b) Resnet-18 tested on Imagenet.

does not go lower afterwards). The accuracy of Resnet-18 drops even more sharply, from 64.9% to 4.1% under 0.1% fault rate. It reaches the random guess accuracy for Imagenet (i.e., 0.1%) once the defect rate elevates to 0.5%. These results are consistent with the error rate v.s. DNN accuracy sensitivity findings reported in [35]. These data indicate that utilizing emerging memories in DNN accelerators is not practical unless effective and efficient mitigation of such defects is provided.

## 2.2 Related Work

Previous work on NVM defect tolerance can be divided into *two categories*: the work designed to tolerate defects in non-volatile memory independent of the systems they are deployed in, and the technique specifically designed for DNN accelerators.

One general approach for tolerating defects in NVMs is Error-correcting-pointers (ECPs) [38], which replaces a defective memory line with a clean line and uses a pointer to record the mapping so that accesses to the defective line can be re-directed. This technique requires two memory accesses to obtain the data, one to the original defective line and one to the clean line. Unfortunately, this two-step redirection is not desirable for DNN accelerators that rely on maintaining a regular structure for speeding up multiple convolutional operations in parallel. Moreover, ECP requires custom NVM devices [51] and relies on the availability of extra storage. Specifically, to mitigate one defective cell, it needs one replacement cell, an N-bit pointer to record the address of the original data, and another bit to indicate all the error-correction entries are in use [38]. This is very costly for DNNs with millions of weights, as a small increase in the defect rate would translate into a significant number of faulty weights that require remapping.

DNN-specific defect mitigation techniques typically require the accelerator's defect map to be obtained *a priori*, which can be achieved with existing defect detection and locating schemes [3, 11, 17, 48]. These techniques mainly adopt an offline testing strategy that writes specific values to memory cells, reads them back, and compares the read values to reference voltages to identify stuck-at-faults. The overhead of such testing processes depends on the test granularity

Table 1. Comparison of Defect Mitigation Techniques for NVM-based DNN Accelerators

Technique	Require retraining?	Require extra storage?	Require extra hardware?
ECP [38]	×	√	√
[25, 26, 48, 52]	√	×	×
[19, 21, 47]	×	√	√
[4, 25]	√	×	√
Proposed	×	×	×

and the size of the memory-under-test. For instance, the testing method in [11] performs writes to each memory cell and imposes high overhead. The approaches in [3, 17] reduce testing overhead by simultaneously writing a group of cells. However, such an offline testing strategy is destructive, i.e., it overwrites the original values of the memory cell, and DNN weights need to be reprogrammed after testing. Given the high programming overhead of NVMs, offline testing is not cost-effective enough to be used as on-line fault detection techniques. In comparison, [48] proposes an on-line defect detection method which performs quiescent-voltage comparison for fast and nondestructive testing of large memories, e.g., a  $1024 \times 1024$  RRAM crossbar can be tested within 70 test cycles.

Once the accelerator’s defect map is obtained with the aforementioned offline and online testing methods, DNN-specific defect mitigation techniques retrain the model to silence the effect of the defects [4, 25, 26, 48, 52]. The work in [52] proposes a defect-aware pruning and retraining technique for systolic array-based neural network accelerators. In [4, 25, 26, 48], the device variations in crossbar-based DNN accelerators are compensated during a defect-aware back-propagation process of network training. However, the original training data set may not always be available to the deployment process. Even if the data set is available, retraining would still be time-consuming and requires a considerable amount of extra computing power especially for large DNNs [41, 45]. What is more, as retraining targets a given defect map, it is not a scalable solution in the typical scenario of a DNN trained once in the cloud and deployed onto many accelerators that are heterogeneous and display unique defect maps due to process variation and aging.

Another class of DNN-specific defect mitigation techniques rely on an extra level of redundancy in DNN accelerators to handle defects [19, 21, 47]. In [19, 21], a complex bypassing of faulty units in the systolic array accelerators is proposed, which not only requires an additional set of routers and registers but also imposes significant overhead as an entire column/row of processing elements should be eliminated per fault. The crossbar-based accelerators in [47] map each weight to two memory cells so that the error in one cell can be compensated by tuning the other. This solution comes at a cost of 2X storage overhead and still cannot handle the extreme case of both cells being faulty. The techniques in [4, 25] do not require extra storage, but perform retraining as well as permuting memory crossbar’s columns to mask the stuck-at faults at deployment-time, which require complicated and costly routing elements.

Table 1 summarizes and compares existing defect mitigation techniques for NVM-based DNN accelerators. As shown, both ECP and DNN-specific approaches have limited applicability to resource- and energy-constrained embedded devices as they depend either heavily on performing costly retraining operations or on the availability of expensive hardware or extra storage. In contrast, this paper presents a light-weight defect mitigation technique for embedded devices which can rescue the accelerator’s accuracy without retraining and requires no storage overhead and negligible hardware overhead.

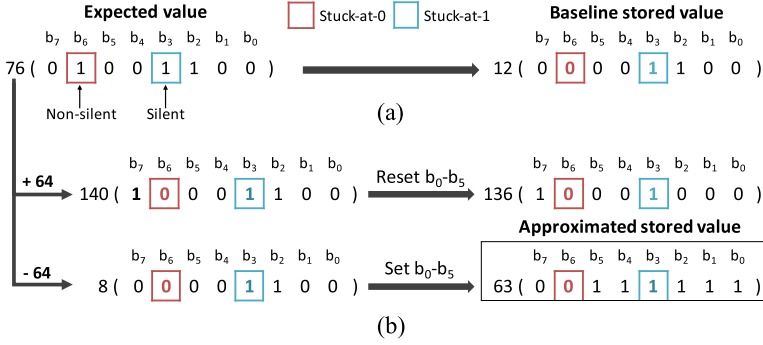


Fig. 2. (a) Large weight value deviation due to the non-silent defect at  $b_6$ . (b) Reduced deviation via Add/Sub Approximation.

### 3 PROPOSED TECHNIQUES

Our work targets DNN accelerators that use traditional SRAM/DRAM and emerging NVM to store the intermediate results and network parameters (i.e., weights), respectively. We assume that SRAM/DRAM memory is equipped with ECC for fault detection and correction, and focus on mitigating the adverse impact of NVM defects that deviate DNN weight values and induce accuracy drop. The location of defects in NVM can be obtained using one of the defect detection and locating techniques [3, 11, 17, 25] reviewed in Section 2.2.

The proposed solutions are developed based on the fundamental observation that a defect deviates a weight value if and only if it is *non-silent* i.e., the value to be stored is not the stuck-at faulty value. Figure 2(a) presents an illustrative example from a 8-bit quantized model. The memory byte has two defective cells, a *stuck-at-1* (*sa1*) cell at  $b_3$  position and a *stuck-at-0* (*sa0*) at  $b_6$ . The defect at  $b_3$  is silent but the one at  $b_6$  is non-silent. As a result of this non-silent fault, the value of 12 (instead of 76) is used for computation. When many weights are influenced by defects, the errors accumulate, leading to a significant drop in the model's inference accuracy.

Based on this observation, we propose three algorithms that *manipulate the flexibility in modifying the fault-free bits* to approximate weight values as close as possible, thus alleviating defect-induced accuracy drop. The problem is formulated below:

**PROBLEM 1.** Given a  $Q$ -bit weight  $W:(b_{Q-1}...b_1b_0)$  and the defect map of the corresponding memory location  $D:(d_{Q-1}...d_1d_0) \mid d \in \{0, 1, X\}$  where each bit has a value of 1 (*sa1*), 0 (*sa0*), or is clean ( $X$ ), find  $W':(b'_{Q-1}...b'_1b'_0) \mid b' \in \{0, 1\}$  such that all the defects in  $D$  are silent when storing  $W'$  in memory and  $W'$  closely approximates  $W$ .

#### 3.1 Add/Sub Approximation

One critical observation is that the impact of a non-silent defect is equal to adding/subtracting 1 to/from the corresponding bit in the weight value. Such deviation can be maximally compensated by altering the lower-order fault-free bits to 0/1. Figure 2(b) shows how the add/sub approximation successfully reduces the weight value deviation compared to Figure 2(a). Since the leftmost non-silent defect is the *sa0* fault at  $b_6$ , two approximated values are created by adding/subtracting 64 to/from 76 and then resetting/setting the fault-free bits in  $b_0 - b_5$  to 0/1. Between the two created values, 63 is closer to the original weight value of 76 and is hence selected to be written to memory.

A further examination shows that the decision on whether to add or subtract 1 can be made without computing both approximated values. Specifically, as Algorithm 1 shows, one can

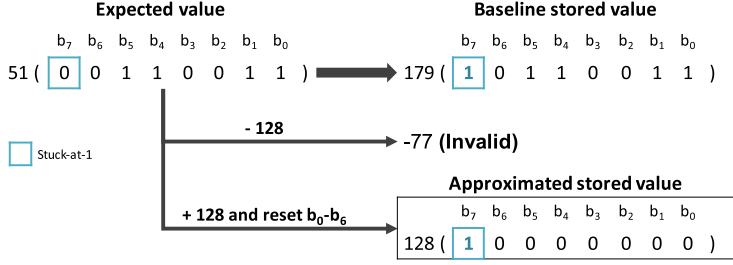


Fig. 3. Add/Sub Approximation ineffectiveness for approximating a weight with *sa1* fault at  $b_7$ .

---

### ALGORITHM 1: Add/Sub Approximation

---

- 1:  $k \leftarrow$  index of the leftmost non-silent defect in  $W$ ,  $0 \leq k \leq Q - 1$
  - 2: **if**  $b_{k-1} = 1$  **then**
  - 3:      $W' \leftarrow W + 2^k$
  - 4:      $b'_j \leftarrow 0$  **for**  $j \in [k - 1, 0]$
  - 5: **else**
  - 6:      $W' \leftarrow W - 2^k$
  - 7:      $b'_j \leftarrow 1$  **for**  $j \in [k - 1, 0]$
  - 8: **if**  $W'$  is invalid **then**
  - 9:      $W' \leftarrow$  approximate  $W$  using the other option
- 

approximate a non-silent fault at bit position  $b_k$  based on the value of  $b_{k-1}$ : if  $b_{k-1}$  is a 1, then adding 1 to  $b_k$  and resetting all the lower-order bits to 0 leads to a closer approximation, while the subtracting-1 option is better if  $b_{k-1}$  is a 0. This property can be observed in Figure 2(b) as well: as  $b_5$  is a 0, the closer approximation is to subtract 1 from  $b_6$  and setting the fault-free bits in  $b_0 - b_5$  to 1. This Add/Sub Approximation algorithm is formally presented in Algorithm 1 (lines 1–7).

The Add/Sub Approximation algorithm works in most cases with the following three exceptions: (1) if the approximated value falls outside the quantized weight value range, (2) if add/subtract operation needs to change the value of higher-order silent faults, or (3) if  $b_{k-1}$  is defective. In these cases, the current add/subtract option does not generate a valid approximation, and the algorithm approximates the weight value via the subtract/add operation instead (lines 8–9 of Algorithm 1). Figure 3 shows an example of an invalid approximation for an 8-bit weight value of 51 wherein  $b_7$  is *sa1* and causes a deviation of 128 (179-51). Since the value of  $b_6$  is 0, the algorithm chooses to subtract 1 from the defective bit which, however, creates an invalid out-of-range result<sup>3</sup> of -77. As a result, the algorithm has no choice but to approximate the value by adding 1 to  $b_7$  which creates the approximated value of 128. This example also shows the *major drawback* of Add/Sub Approximation: although the algorithm is capable of reducing the deviation in weight values, a non-silent *sa1/sa0* fault at the high-order bits of a small/large weight value can still lead to a large deviation.

### 3.2 Flipping-based Approximation

If a non-silent defect locates in a high-order bit of the weight, its impact cannot be effectively mitigated via manipulating the lower-order bits. Instead, our observation is that if all the bits of

<sup>3</sup>The example assumes that the quantized weights are unsigned numbers, however, the algorithm can be applied to 2's complement numbers as well.

---

**ALGORITHM 2: LSB Approximation**

---

```
1:  $W' \leftarrow W$ 
2:  $Flipped \leftarrow 0$ 
3: if LSB is fault-free and leftmost defect is non-silent then
4:    $b'_j = 1 - b_j$  for  $j \in [Q - 1, 0]$ 
5:    $Flipped \leftarrow 1$ 
6: else if LSB is a sa1 fault then
7:    $b'_j \leftarrow 1 - b_j$  for  $j \in [Q - 1, 1]$ 
8:    $Flipped \leftarrow 1$ 
9: if still non-silent defects remaining in  $W'$  then
10:   Use Add/Sub to approximate the leftmost remaining fault
11:  $b'_0 \leftarrow Flipped$ 
```

---

the weight are flipped and then stored in the memory, the non-silent defect will become silent. While the idea, sounds simple, the major challenge is to cost-effectively encode the information of whether a weight has been flipped or not, so that it can be conditionally flipped back at run-time before being used in computation. We propose two approaches that encode the conditionally flipping information directly into the network weight values *without incurring any extra storage*, namely, *LSB Approximation* and *Count-One (C-1) Approximation*.

**3.2.1 LSB Approximation.** The LSB Approximation utilizes the weight's Least Significant Bit (LSB) to indicate weight-flips.<sup>4</sup> Algorithm 2 shows the approximation process. It assigns a *Flipped* flag to each weight to indicate whether it has gone through the weight-flip operation. Initially the flag is set to 0 (line 2). If the leftmost defect is non-silent, it flips the weight value bit by bit and sets the flag to 1 (lines 3–5). The remaining non-silent defect, if any, is mitigated via Add/Sub Approximation (lines 9–10). At the end, the value of LSB ( $b_0$ ) is matched with the value of the flag (line 11). The obtained weight  $W'$  is then stored to memory. At run-time, the LSB approximation technique reads the weight value and checks its LSB. The value is then flipped back if and only if  $LSB = 1$ .

Figure 4(a) demonstrates the ability of LSB Approximation in minimizing weight deviation for the highest-ordered *sa1* cell in the example of Figure 3. Here,  $b_7$  is the leftmost defect and because it is non-silent, the weight is flipped and the *Flipped* flag is set. Since there is no more defect, the algorithm terminates by setting  $b_0$  to 1 which results in the value of 205 being stored in memory. At run-time, the system reads the weight and checks its LSB. Since  $b_0$  is 1, it flips all the weight bits and as a result, the value of 50 is used in computation which creates a minimum deviation of 1 from the expected weight value of 51. Figure 4(b) shows the process of applying both LSB Approximation and Add/Sub Approximation to the example in Figure 2, which has more than one stuck-at faults in a weight. Integrating the two approximation approaches together reduces the weight deviation from 13 (76-63) in Figure 2(b) to 6 (76-70).

It is important to note that LSB Approximation is uniformly applied to all the weights in the network, regardless of whether they are faulty or not. This may lead to a minimum deviation in the fault-free weight values as the LSBs of all fault-free weights are set to 0. While this minimum deviation has probably negligible impact,<sup>5</sup> a more critical drawback is the case when the LSB itself

---

<sup>4</sup>The technique can also assign an extra bit per weight, however, here an existing bit is used to minimize the hardware overhead.

<sup>5</sup>Such deviation can be avoided if an extra bit is used to encode flip/no-flip information per weight, or if the weights are quantized to be 1-bit shorter and the LSB is not used during computation but only encodes flip/non-flip information.

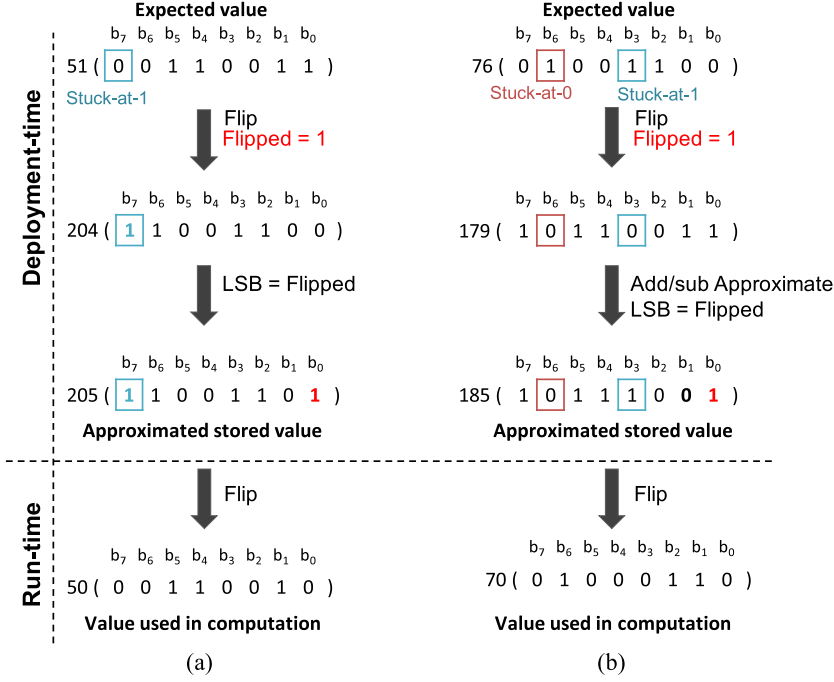


Fig. 4. (a) LSB Approximation successfully silences the *sa1* fault in  $b_7$ , (b) LSB Approximation used together with Add/Sub Approximation to reduce the deviation.

is defective. As shown in Algorithm 2 (lines 6–8), the weight has to be flipped/not-flipped according to the LSB’s stuck-at value which may contradict the correction desired by the leftmost non-silent defect. Figure 5(b) shows an example of this case where  $b_0$  and  $b_7$  are respectively non-silent and silent *sa1* faults. Without LSB Approximation, the value stored in memory would be 203. With LSB Approximation, however, the weight has to be flipped since  $b_0$  is *sa1*. This makes the defect at  $b_7$  non-silent and thus the algorithm will use Add/Sub Approximation to mitigate it. The final weight value stored in memory is 129 and the value used in computation is 126, which is a larger deviation from the expected value. This example shows that LSB approximation is limited by the probability of conflicts caused by defective LSBs. Even if an extra bit is used to encode flip/no-flip information per weight (or if the weights are quantized to be 1-bit shorter and the LSB is used to encode flip/no-flip information), such conflicts still may occur since the extra bit can be faulty as well.

**3.2.2 Count-One (C-1) Approximation.** Instead of relying on a single bit that may be faulty, C-1 Approximation encodes the existence/absence of weight-flip via forming an odd/even number of 1’s among all of the bits in a weight. This property is formed by selectively modifying the lowest non-faulty bit in the weight, thus minimizing the possibility of conflicts. Algorithm 3 shows the encoding process, which starts by assigning a *1-Count* flag, that is initiated to “even” (line 2). Then, the leftmost defect is examined. If it is non-silent, the weight is flipped and the flag is changed to “odd” (lines 3–5). The remaining non-silent defects, if any, are mitigated via Add/Sub Approximation (lines 6–7). At the end, the algorithm ensures that the 1’s count property holds, by checking the number of 1’s in the weight and the *1-Count* flag value. If they are inconsistent, (i.e., the flag is odd/even while there are even/odd number of 1’s in the weight value), the rightmost

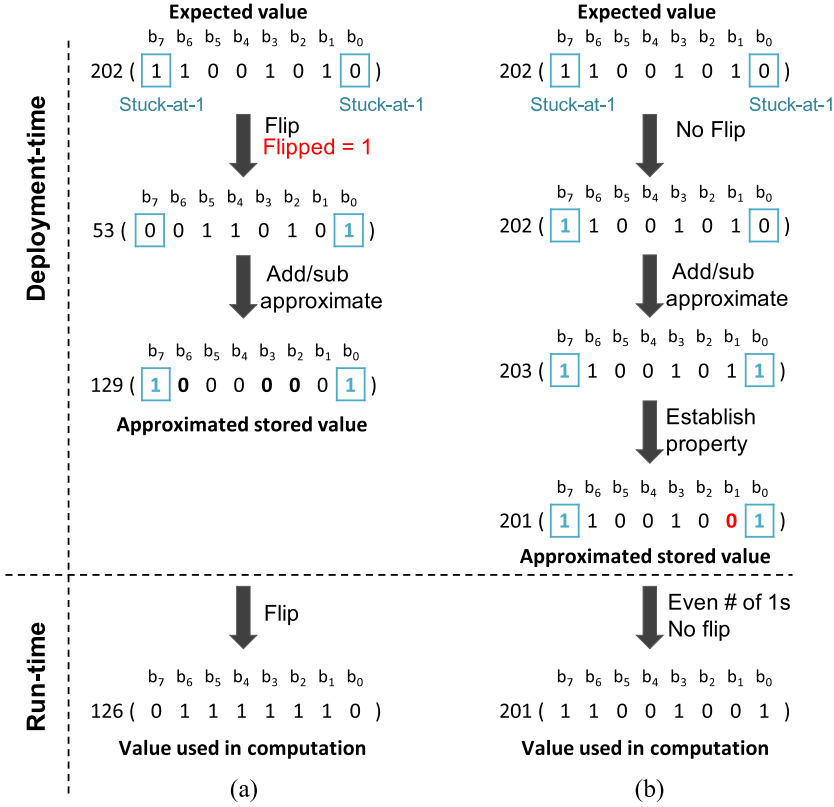


Fig. 5. (a) Conflicts in the LSB Approximation when LSB is defective, (b) Conflict-free C-1 Approximation.

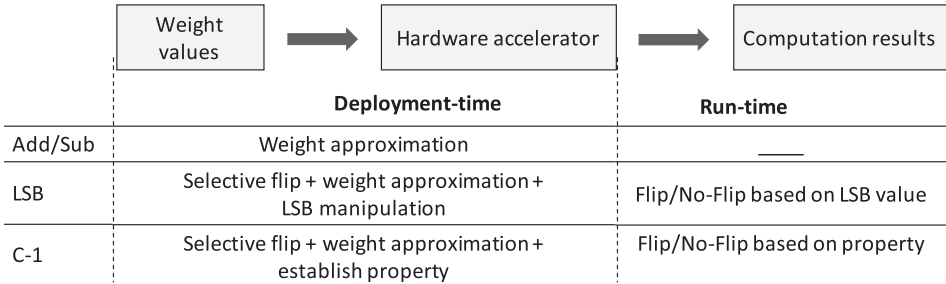


Fig. 6. Procedural overview of the proposed techniques.

fault-free bit of the weight is flipped to enforce the property (lines 8–9). At run-time, the system reads the weight value, counts the number of 1’s in it, and flips it bit-by-bit if any only if the count is odd.

Figure 5(b) shows how the C-1 Approximation tolerates the two *sa1* defects and reduces the deviation from 76 ( $202 - 126$ ) in Figure 5(a) to 1 ( $202 - 201$ ). In the first step, the weight is not flipped as its leftmost defect at  $b_7$  is silent. Next, Add/Sub approximation is used for the  $b_0$  defect, which leads to an odd number of 1’s in the weight. Then, to make the C-1 property hold, the

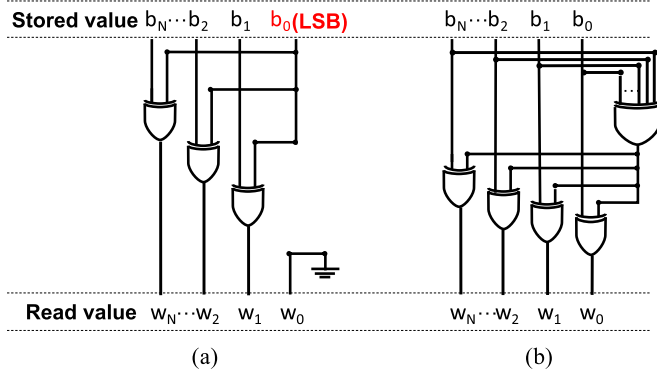


Fig. 7. (a) LSB Approximation read circuitry, (b) C-1 Approximation read circuitry.

---

### ALGORITHM 3: C-1 Approximation

---

- 1:  $W' \leftarrow W$
  - 2:  $1\text{-Count} \leftarrow \text{even}$
  - 3: **if** leftmost defect is non-silent **then**
  - 4:    $b'_j \leftarrow 1 - b_j$  **for**  $j \in [Q - 1, 0]$
  - 5:    $1\text{-Count} \leftarrow \text{odd}$
  - 6: **if** still non-silent defects remaining in  $W'$  **then**
  - 7:   Use Add/Sub to approximate the leftmost remaining fault
  - 8: **if** ( $1\text{-Count} = \text{odd}$  **and** number of 1's in  $W'$  is even) **or** ( $1\text{-Count} = \text{even}$  **and** number of 1's in  $W'$  is odd) **then**
  - 9:   Flip the rightmost fault-free bit of  $W'$
- 

rightmost fault-free bit  $b_1$  is flipped, generating the value of 201 for memory storage. At run-time, the system reads the stored weight value and since it has an even number of 1's, the system does not flip it and uses 201 in computation.

The C-1 Approximation is expected to outperform LSB Approximation by minimizing the possibility of conflicts – as long as there are fault-free bits, the weight can be well-approximated. Since the technique relies on forming the C-1 property for all the weights, healing faulty weights also comes at the cost of creating minor deviations in fault-free weights where the total number of 1's is not even. As will be shown in Section 4, such minor deviations will not overpass the benefit obtained from the fine approximation of faulty weights.

### 3.3 Run-time Support

So far, we have demonstrated the ability of the three approximation algorithms in mitigating the impact of permanent faults in accelerator memory. It is important to note that these algorithms impose **zero** storage overhead and require minimal run-time support. Figure 6 summarizes the steps taken at both deployment-time and run-time. Since deployment is a one-time process which loads the DNN model's parameters to the NVM memory, the proposed weight approximation algorithms only bring in a one-time offline overhead. In terms of run-time support, the Add/Sub Approximation requires no support as it relies entirely on deployment-time modification of the weights, while the LSB and C-1 Approximation require selective weight bit flipping, which can be achieved with very limited hardware added to the weight read circuitry, as shown in Figure 7.

Table 2. Target Datasets and Evaluated DNN Models' Specifications, Accuracy, and Quantization

Dataset	DNN	Layers	Weights	Accuracy	Accuracy after quantization		
CIFAR-10	CifarNet	18	$1.25 \times 10^6$	78.39%	# of bits	8	16
					Accuracy	74.78%	78.28%
	VGG-16	23	$15 \times 10^6$	93.32%	# of bits	8	16
					Accuracy	89.85%	93.32%
Imagenet	Resnet-18	8	$11 \times 10^6$	64.97%	# of bits	16	
					Accuracy	64.92%	
MNIST	LeNet	8	$1.20 \times 10^6$	99.15%	# of bits	4	
					Accuracy	99.11%	

To conditionally flip an  $N$ -bit weight in LSB Approximation, all bits of the stored weight value are XOR-ed with the LSB (as a result, the LSB of the read value is always 0), which can be achieved with  $N - 1$  2-input XOR gates. Whereas, computing whether the number of 1's is even or odd in the C-1 Approximation requires one  $N$ -input XOR gate as well as  $N$  2-input XOR gates to perform conditional weight-flip. Such hardware and timing overhead brought by these techniques is negligible.

## 4 EXPERIMENTAL EVALUATIONS

### 4.1 Methodology

*4.1.1 Benchmarks.* The approximation algorithms proposed in this work are generally applicable to any DNN workloads. To demonstrate its wide applicability, this section evaluates the proposed techniques under three well-known datasets and four diverse DNN models – as representatives of standard applications deployed on neural network accelerators. Table 2 shows the data sets and the selected DNN structures, along with their total number of layers, total number of weights, original accuracy (of floating-point models), and accuracy after quantization. The evaluated datasets include two object recognition sets, CIFAR-10 [20] and Imagenet [7], as well as one handwritten digit classification set, MNIST [22]. CIFAR-10 consists of 60K  $32 \times 32$  color images (50K training and 10K testing) that are classified into 10 classes; Imagenet dataset is the largest dataset with 1.35M (1.2M training and 150K testing)  $500 \times 375$  color images in 1000 classes; MNIST has 70K  $28 \times 28$  grey-level images (60K training and 10K testing) that are classified in 10 classes. As our target systems are battery- and resource-constrained devices, our experiments are performed on quantized DNN structures. Resnet-18 weights are quantized in 16 bits to retain maximum accuracy (as close as possible to the original floating-point model), while LeNet weights are quantized in 4 bits. CifarNet and VGG-16 are evaluated under both 8-bit and 16-bit quantization levels to further evaluate the impact of different quantization levels on the effectiveness of our solution.

*4.1.2 Fault Injection Experiments.* All the fault injection and accuracy assessment processes are implemented in Keras [5]. All the quantized DNN models are evaluated under six different defect rates ranging from 0.1% to 5%, where  $x\%$  defect rate means that each memory cell has  $x\%$  probability to be a defect. For each combination of defect rate and quantization level, 20 different defect maps are randomly generated.

One important consideration in fault injection studies is the distribution of different fault types. While in some NVMs such as PCM, the chance of stuck-at-0 (sa0) vs stuck-at-1 (sa1) faults is 50%–50%, in RRAM the probability of sa1 fault is 4X higher than sa0 as reported in [3]. Since in most DNN models the majority of weight values are close to zero, the impact of sa0 faults is less significant than sa1 faults. In other words, the asymmetric distribution with higher rate of sa1 faults

Table 3. Average Ratio of Network Weights With 1 and 2+ Non-silent Faults Per Weight, e.g., 0.195% is the Percentage of Network Weights with 1 Non-silent Fault when the Defect Rate is 0.1%

Quantization level	Non-silent fault per weight	Defect rate (%)					
		0.1%	0.2%	0.5%	1%	2%	5%
4-bit	1	0.195%	0.389%	0.967%	1.925%	3.791%	6.608%
	2+	0.000%	0.001%	0.003%	0.013%	0.054%	1.571%
8-bit	1	0.390%	0.777%	1.923%	3.782%	7.315%	16.506%
	2+	0.001%	0.003%	0.016%	0.065%	0.253%	1.491%
16-bit	1	0.781%	1.563%	3.785%	7.335%	13.619%	27.033%
	2+	0.003%	0.011%	0.071%	0.279%	1.065%	6.020%

is more harmful to network accuracy and more difficult to recover from. Targeting the worst case, our fault injection experiments adopt this asymmetric fault distribution, and the injected rates of sa1 vs sa0 faults are 4:1.

To ensure the fairness of our fault injection setup, we performed detailed profiling of the generated defect maps. The obtained data confirm that faults are uniformly distributed across all bit positions (from MSB to LSB). Moreover, the generated defect maps cover a diverse set of fault scenarios including various number of non-silent defects per weight. Table 3 reports the average ratio of network weights with 1 and 2 (or more) non-silent faults. The data are averaged across different models with the same quantization level, as the probability of faults per weight is only affected by the quantization level and the defect rate,<sup>6</sup> not by the network structure or weight count. The data shows that while the majority of network weights are fault-free, the probability of non-silent faults per weight increases super-linearly as the quantization level increases. At a high defect rate, the chance of having two or more non-silent faults in a single weight becomes significant, especially for the 16-bit models.

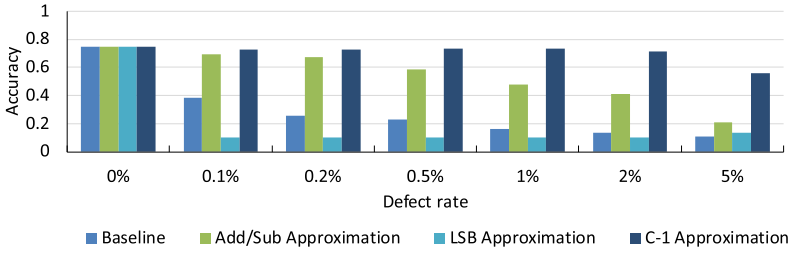
## 4.2 Results

**4.2.1 Accuracy Improvement.** Our first set of experiments evaluate the efficacy of the three proposed algorithms in rescuing the target DNN’s accuracy. Figures 8(a)-(b) and Figures 8(c)-(d) respectively display the average accuracy of CifarNet and VGG-16 under two quantization-levels and six defect rates. One can observe that both DNN structures are highly sensitive to the accumulation of faults as their baseline accuracy drops sharply even under the lowest defect rate of 0.1%. At 5% defect rate, the accuracy of CifarNet descends to 10% which is equivalent to the random guess accuracy for Cifar-10 (as it has 10 classes), while the accuracy of VGG-16 drops to 10% even at 0.1% defect rate.

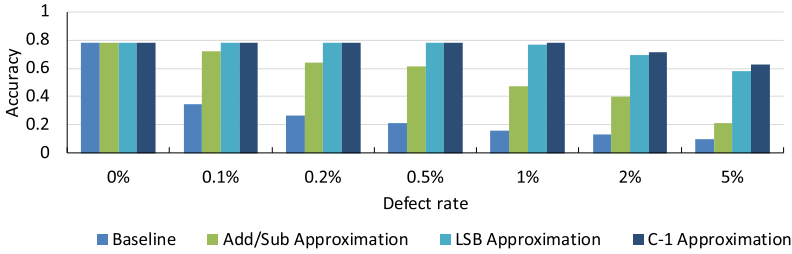
Among the proposed techniques, C-1 Approximation consistently outperforms the other two in both 8- and 16-bit CifarNet and VGG models. Generally, as the defect rate increases, the accuracy of all the models drops across different approximation techniques. However, the higher the defect rate, the better the C-1 Approximation is compared to the other two. Even at 2% defect rate, C-1 Approximation can deliver an accuracy of more than 70% for both 8- and 16-bit CifarNet models, and around 90% for 8- and 16-bit VGG-16 models.

In comparison, Add/Sub Approximation, which requires no run-time support, is a favorable choice for mitigating defect rates of 0.2% and lower. However, its performance gradually degrades as the defect rate increases, and becomes less effective for all CifarNet and VGG-16 models once

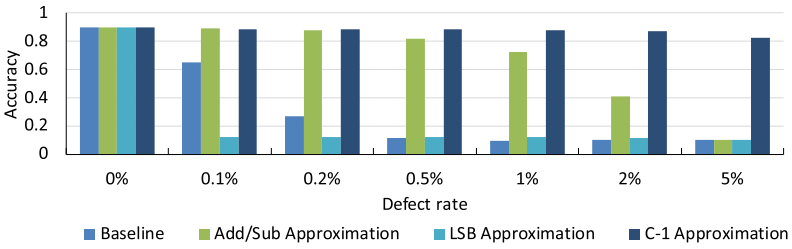
<sup>6</sup>For a quantization level of  $n$ -bit and a fault rate of  $x\%$ , the probability for an  $n$ -bit weight to have at least one faulty bit is  $1 - (1 - x\%)^n$ . The data reported in Table 3 are lower than the theoretical values as only non-silent faults are counted.



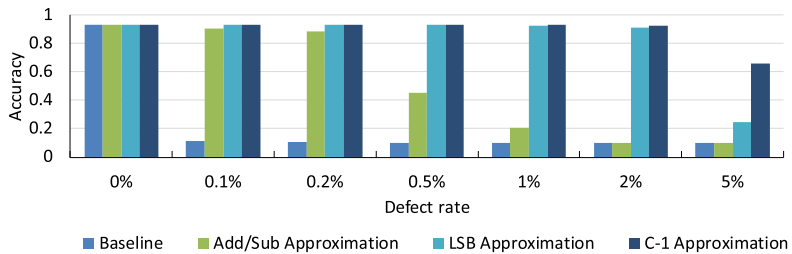
(a) 8-bit CifarNet



(b) 16-bit CifarNet



(c) 8-bit VGG-16



(d) 16-bit VGG-16

Fig. 8. Impact of the proposed techniques on the accuracy of CifarNet and VGG-16.

the defect rate is increased beyond 0.2%. This is because, as we discussed in Section 3.1 and illustrated in Figure 3, the impact of a non-silent fault in a high-order bit position cannot be effectively mitigated via manipulating the lower-order bits in the weight. The probability of this case solely

Table 4. Impact of the Proposed Techniques on the Accuracy of 16-bit Resnet-18

Technique	Fault rate(%)						
	0	0.1	0.2	0.5	1	2	5
Baseline	64.92	4.05	0.24	0.10	0.10	0.10	0.10
Add/Sub Approximation	64.92	63.55	62.25	55.48	30.07	0.35	0.10
LSB Approximation	64.92	64.94	64.90	64.84	64.47	62.94	40.57
C-1 Approximation	64.92	64.93	64.90	64.82	64.54	63.33	43.34

depends on the defect rate, not the quantization level. As a result, for the same network model, Add/Sub Approximation behaves similarly across different quantization levels.

In terms of LSB Approximation, it displays quite diverse levels of effectiveness for 8- and 16-bit models, as shown in Figure 8. For 8-bit models, LSB Approximation is consistently outperformed by the other two techniques, while for 16-bit models it shows higher effectiveness, close to C-1 Approximation. The reason for this diverse behavior is that LSB Approximation needs to use one bit per weight to record the flipping status which actually reduces the quantization level from  $N$  to  $N - 1$ . For CifarNet and VGG-16, however, a minimum quantization level of 8-bit is required to maintain acceptable accuracy. As a result, LSB Approximation causes a significant accuracy drop for 8-bit models but not for the 16-bit versions. The figure also shows that for 16-bit networks, the only case that LSB Approximation is not able to maintain high accuracy is VGG-16 under 5% defect rate. This is because the 5% defect rate in 16-bit networks causes a high ratio of weights to suffer two or more non-silent faults, as one can observe from Table 3. This in turn leads to a higher chance of conflicts (such as the one shown in Figure 5(a)) when LSB Approximation is used. Note that although the ratio of weights with two or more faults is similar in VGG-16 and CifarNet, VGG-16 ends up experiencing more conflicts simply because it is a larger network.

The experimental results on the 16-bit Resnet-18 model show similar trends. Table 4 presents the accuracy results of Resnet-18 on the Imagenet dataset. Compared to the previous two network models studied on Cifar-10, Resnet-18 is even more sensitive to defects. Its accuracy drops sharply to  $\sim 4\%$  with only 0.1% defect rate and quickly reaches the random guess accuracy (i.e. 0.1% for Imagenet as it has 1000 classes) at 0.5% defect rate. Despite such a high-level of fault sensitivity, both C-1 Approximation and LSB Approximation can successfully rescue the model accuracy. Even under a high defect rate of 2%, C-1 Approximation can improve the accuracy to 63.33% while LSB Approximation offers a slightly lower performance, achieving 62.94% accuracy. In comparison, Add/Sub Approximation’s performance is very close to the flipping-based techniques when the defect rate is 0.2% or lower, while its effectiveness reduces for 1% or higher fault rates. These data show that Add/Sub Approximation is a good choice for resource-constrained devices with 0.2% or lower defect rates, while in the case of higher fault rates either of the flipping-based approximations can be selected to rescue the model’s accuracy.

We also performed accuracy study on 4-bit LeNet, and the results are reported in Table 5. The data show that LeNet is intrinsically more resilient (i.e., MNIST is more recognizable) – even under a 5% defect rate, the baseline accuracy of 4-bit LeNet only drops by 3.12%. Among the three approximation techniques, C-1 is still the most effective in general. LSB approximation does not perform well, as it ends up reducing the quantization level to 3-bit, which is insufficient for LeNet to maintain a high accuracy. Finally, Add/Sub Approximation offers comparable and sometimes even better results, especially at lower defect rates. Since Add/Sub Approximation is more affordable, it is therefore the recommended defect mitigation technique for LeNet accelerators.

Overall, the results of CifarNet, VGG-16, and LeNet show that the proposed C-1 Approximation is the most effective approximation method across all the networks, both low and high fault rates,

Table 5. Impact of the Proposed Techniques on the Accuracy of 4-bit LeNet

Technique	Fault rate(%)						
	0	0.1	0.2	0.5	1	2	5
Baseline	99.11	98.64	99.09	99.05	98.97	98.74	95.99
Add/Sub Approximation	99.11	99.11	99.11	99.09	99.09	99.06	98.90
LSB Approximation	99.11	18.88	18.88	18.88	18.83	18.90	18.94
C-1 Approximation	99.11	99.09	99.09	99.10	99.07	99.07	98.81

and different quantization levels. If the defect rate in the target system is expected to be low or the network model is intrinsically more resilient (such as LeNet for simple MNIST dataset), then the Add/Sub Approximation which is more affordable is a good alternative.

**4.2.2 Deviation Reduction.** After demonstrating the capability of preserving accuracy, we performed an in-depth study to understand why the models’ accuracy can be rescued by the proposed techniques. Specifically, we evaluated the distribution of deviations in weight values before and after applying each approximation technique. Our study focuses on 16-bit CifarNet for a randomly selected defect map of 5% fault rate wherein the three approaches show more distinct behaviors.

Figure 9 presents the histograms of weight deviations for the selected experiment. The x-axis shows the base-2 logarithm of deviation in faulty weights (i.e.  $x$  means the deviation in weight value falls in the range of  $[2^x, 2^{x+1}]$ ) and the Y-axis shows the total number of weights whose deviation falls into the corresponding range. The total number of deviated weights and the obtained accuracy are shown at the top of each figure.

The deviation distribution of the baseline (no approximation) is shown in Figure 9(a). About 8% (96031) of the weights are deviated by the non-silent defects. Many of the non-silent defects occur in the MSB (i.e., the 15-bit bar), and the DNN accuracy drops to 10%. Note that the uneven distribution of deviation across the x-axis is due to the fact that most of DNN weight values center around 0, which is represented as  $0 \times 7FFF$  in 16-bit unsigned numbers. As 80% of the injected faults are stuck-at-1 faults, a large deviation is shown in the 15-bit bar but not the 14-bit bar. Figures 9(b)-(d) illustrate the change in this distribution after applying the three approximation techniques. The Add/Sub Approximation changes the histogram shape by reducing the deviations in the 15-bit bar and generating more deviations in other bars. However, the overall deviation in some weights is still considerably large. The accuracy is improved to 26.11% which is still much lower than the original accuracy. In comparison, both LSB and C-1 Approximations result in a way more significant change. As both approaches mask the leftmost non-silent faults, the deviation originally in the 15-bit bar is eliminated completely. On the other hand, both approximations deviate many weights that are originally fault-free as they need to selectively flip the LSB or the rightmost non-faulty bit. As a result, the total number of deviated weights is raised to about 50% (627503 in LSB and 628669 in C-1) of the total weights. However, most of these introduced deviations are minimal (i.e., in the 1-bit bar) and thus do not suppress the significant benefit of eliminating non-silent MSB errors. As a result, LSB Approximation raises the accuracy to 61.42% and C-1 Approximation, with fewer weights falling into the higher deviation range, achieves 75.32% accuracy, which is very close to the original accuracy of 78.28% of 16-bit CifarNet. Overall, this study demonstrates that the proposed techniques improve accuracy by effectively mitigating the large deviations caused by defects in the higher-order bits of DNN weights.

**4.2.3 Comparison Against Retraining-based Scheme.** As reviewed in Section 2.2, the majority of existing defect mitigation techniques rely on retraining the model as part of their solutions to

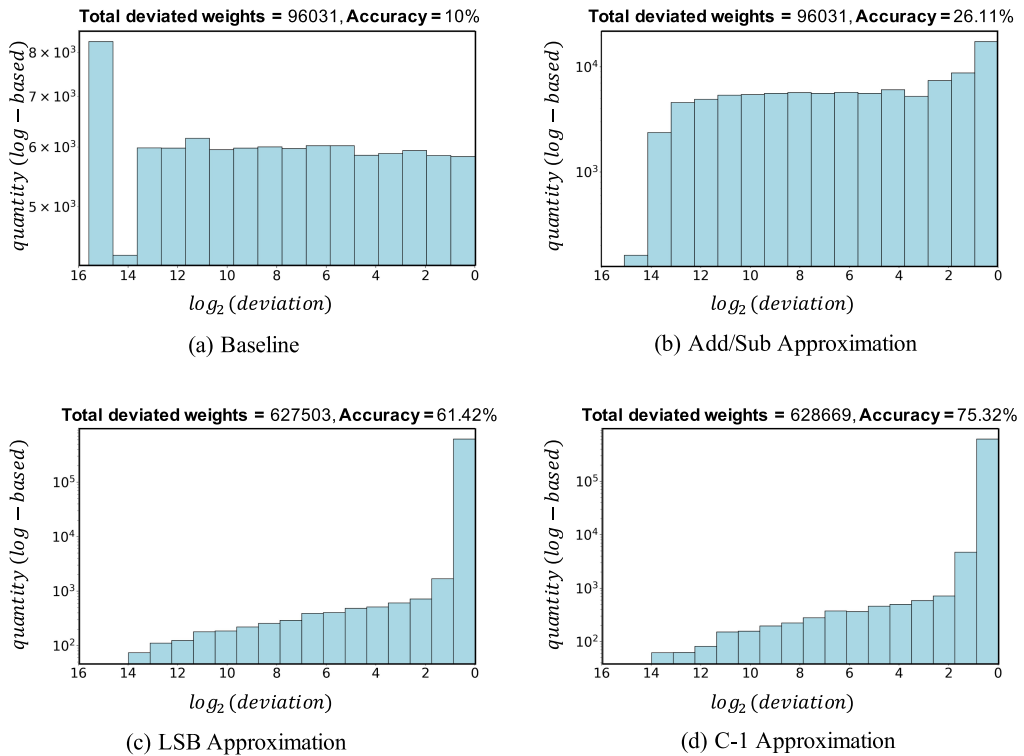
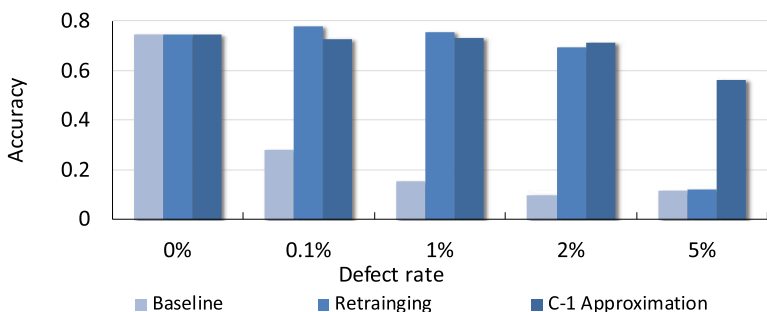


Fig. 9. Distribution of deviation in 16-bit CifarNet weights.

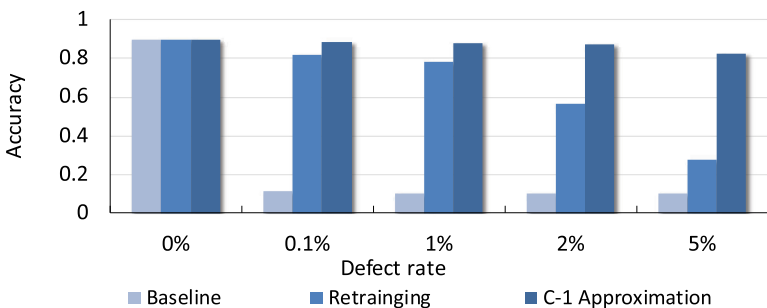
silent the effect of defects. In this study, we compare our proposed C-1 Approximation technique with the defect-aware retraining process proposed in [48]. The goal is to check whether our proposed technique can rescue the model's accuracy to a similar level of retraining-based approaches without the costly and non-scalable retraining.

As retraining is non-scalable and dependent on defect maps, to simplify the process, we only retrain one random defect map under each fault rate for 8-bit CifarNet, 8-bit VGG-16, and 4-bit LeNet models. The post-retraining accuracy compared with the accuracy achieved by C-1 approximation for the corresponding defect map is reported in Figure 10. As can be seen, the retraining process is effective in recovering the model accuracy in general. However, once the defect rate increases beyond a specific point (e.g., 2% for CifarNet and 1% for VGG-16), the effectiveness of retraining drops significantly. In comparison, the C-1 Approximation can rescue the model accuracy to a similar level under both low and medium fault rates. Under a high defect rate such as 5%, it outperforms retraining significantly for both CifarNet and VGG-16. For LeNet on the simple MNIST dataset, due to its high intrinsic resilience, both schemes are capable of recovering the accuracy.

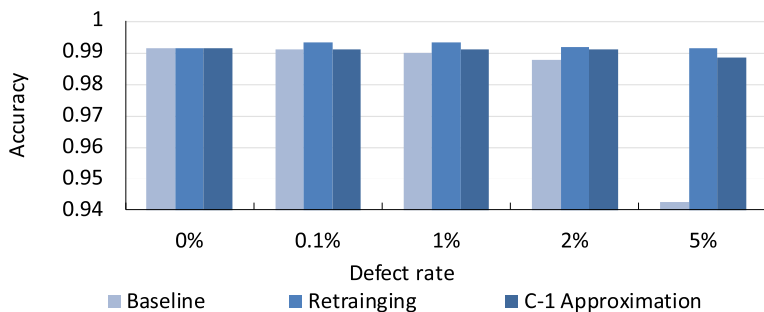
Overall, the results show that our solution outperforms the retraining solution even better as the dataset and DNN become larger in terms of both accuracy (especially at a higher defect rates) and the training cost. Please note that retraining of larger networks can be much more costly than smaller ones. Hence, we believe that our technique is a better fit for recourse-constrained embedded devices as it requires no retraining and imposes zero storage overhead and negligible hardware overhead.



(a) 8-bit CifarNet



(b) 8-bit VGG-16



(c) 4-bit LeNet

Fig. 10. Comparison of the proposed C-1 Approximation against retraining.

## 5 CONCLUSION

In this paper, we have proposed a set of retraining-free and low-cost solutions to tolerate permanent defects in NVM memories of DNN accelerators. Without imposing any storage overhead, the proposed solutions manipulate the flexibility in altering fault-free bits to minimize the impact of defects on weight values. By utilizing the defective cells but mitigating their adverse impact, these techniques allow for larger DNNs to be deployed on resource-constrained devices quickly. Their negligible run-time overhead furthermore makes them excellent candidates for energy-constrained platforms. These techniques have been evaluated on four DNN structures for three popular datasets under different quantization levels and fault rates, and the promising

results demonstrate their great potential in rescuing the accelerators' accuracy compared with existing solutions even in the presence of high defect rates. Future work will focus on extending this work to cover NVM-specific transient faults by incorporating runtime error detection and correction capability.

## REFERENCES

- [1] K. Beckmann, J. Holt, H. Manem, J. Van Nostrand, and N. C. Cady. 2016. Nanoscale Hafnium Oxide RRAM devices exhibit pulse dependent behavior and multi-level resistance capability. *Mrs Advances* 1, 49 (2016), 3355–3360.
- [2] G. W. Burr, R. M. Shelby, S. Sidler, N. C. Di, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, et al. 2015. Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Transactions on Electron Devices* 62, 11 (2015), 3498–3507.
- [3] C. Chen, H. Shih, C. Wu, C. Lin, P. Chiu, S. Sheu, and F. T. Chen. 2014. RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Trans. Comput.* 64, 1 (2014), 180–190.
- [4] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang. 2017. Accelerator-friendly Neural-network Training: Learning variations and Defects in RRAM Crossbar. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. 19–24.
- [5] F. Chollet et al. 2015. Keras. <https://keras.io>. (2015).
- [6] N. DeBardleben, S. Blanchard, V. Sridharan, S. Gurumurthi, J. Stearley, K. Ferreira, and J. Shalf. 2014. Extra Bits on SRAM and DRAM Errors—More Data From the field. In *IEEE Workshop on Silicon Errors in Logic-System Effects (SELSE)*.
- [7] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. <https://doi.org/10.1109/CVPR.2009.5206848>
- [8] G. Dhiman, R. Ayoub, and T. Rosing. 2009. PDRAM: A Hybrid PRAM and DRAM main memory system. In *2009 46th ACM/IEEE Design Automation Conference*. 664–669.
- [9] M. Donato, L. Pentecost, D. Brooks, and G. Wei. 2019. MEMTI: Optimizing On-chip Nonvolatile storage for visual multitask inference at the edge. *IEEE Micro* 39, 6 (2019), 73–81.
- [10] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis. 2017. TETRIS: Scalable and efficient neural network acceleration with 3D memory. *SIGARCH Comput. Archit. News* 45, 1 (2017), 751–764.
- [11] A. Van De Goor and Y. Zorian. 1994. Effective march algorithms for testing single-order addressed memories. *Journal of Electronic Testing* 5, 4 (1994), 337–345.
- [12] M. Gottscho, M. Shoaib, S. Govindan, B. Sharma, D. Wang, and P. Gupta. 2017. Measuring the impact of memory errors on application performance. *IEEE Computer Architecture Letters* 16, 1 (2017), 51–55.
- [13] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras. 2019. Terminal brain damage: exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA, 497–514.
- [14] L. Jiang, M. Kim, W. Wen, and D. Wang. 2017. XNOR-POP: A processing-in-memory architecture for binary convolutional neural networks in wide-IO2 DRAMs. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6.
- [15] M. R. Jokar, M. Arjomand, and H. Sarbazi-Azad. 2016. Sequoia: A high-endurance NVM-based cache architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 3 (2016), 954–967.
- [16] N. P. Jouppi, C. Young, N. Patil, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. 1–12.
- [17] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu. 2015. Modeling, detection, and diagnosis of faults in multilevel memristor memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 34, 5 (2015), 822–834.
- [18] H. A. Khouzani, F. S. Hosseini, and C. Yang. 2016. Segment and conflict aware page allocation and migration in DRAM-PCM hybrid main memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 9 (2016), 1458–1470.
- [19] J. H. Kim and S. M. Reddy. 1989. On the design of fault-tolerant two-dimensional systolic arrays for yield enhancement. *IEEE Trans. Comput.* 38, 4 (1989), 515–525.
- [20] A. Krizhevsky, V. Nair, and G. Hinton. 2010. CIFAR-10 (canadian institute for advanced research). (2010). <http://www.cs.toronto.edu/~kriz/cifar.html>.
- [21] H. T. Kung and M. S. Lam. 1983. Fault-tolerance and two-level pipelining in VLSI systolic arrays.
- [22] Y. LeCun and C. Cortes. 2010. MNIST handwritten digit database. (2010). <http://yann.lecun.com/exdb/mnist/>.
- [23] S. Lee, H. Bahn, and S. H. Noh. 2014. CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures. *IEEE Trans. Comput.* 63, 9 (2014), 2187–2200.

- [24] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie. 2017. DRISA: A DRAM-based reconfigurable in-situ accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 288–301.
- [25] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang. 2015. Vortex: Variation-aware training for memristor X-bar. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.
- [26] C. Liu, M. Hu, J. P. Strachan, and H. Li. 2017. Rescuing memristor-based neuromorphic design with high defects. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.
- [27] Y. Long, X. She, and S. Mukhopadhyay. 2019. Design of Reliable DNN accelerator with unreliable ReRAM. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.
- [28] S. Longofono, D. Kline, R. G. Melhem, and A. K. Jones. 2020. A CASTLE with TOWERs for Reliable, Secure PCM. *IEEE Trans. Comput.* (2020), 1–1.
- [29] Y. Luo and S. Yu. 2020. Accelerating deep neural network in-situ training with non-volatile and volatile memory based hybrid precision synapses. *IEEE Trans. Comput.* 69, 8 (2020), 1113–1127.
- [30] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. 2005. The soft error problem: An architectural perspective. In *11th International Symposium on High-Performance Computer Architecture*. 243–247.
- [31] Y. Pan, P. Ouyang, Y. Zhao, W. Kang, S. Yin, Y. Zhang, W. Zhao, and S. Wei. 2018. A Multilevel Cell STT-MRAM-based computing in-memory accelerator for binary convolutional neural network. *IEEE Transactions on Magnetics* (2018), 1–5.
- [32] Y. Park, D. Shin, S. K. Park, and K. H. Park. 2011. Power-aware memory management for hybrid main memory. In *The 2nd International Conference on Next Generation Information Technology*. 82–85.
- [33] L. Pentecost, M. Donato, B. Reagen, U. Gupta, S. Ma, G. Wei, and D. Brooks. 2019. Maxnvm: Maximizing DNN storage density and inference efficiency with sparse encoding and error mitigation. In *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*. 769–781.
- [34] A. S. Rakin, Z. He, and D. Fan. 2019. Bit-flip Attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE International Conference on Computer Vision*. 1211–1220.
- [35] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G. Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6.
- [36] D. Roberts and P. Nair. 2014. FAULTSIM: A Fast, configurable memory-resilience simulator. In *The Memory Forum: In Conjunction with ISCA*, Vol. 41.
- [37] R. Salkhordeh and H. Asadi. 2016. An operating system level data migration scheme in hybrid DRAM-NVM memory architecture. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 936–941.
- [38] S. Schechter, G. H. Loh, K. Strauss, and D. Burger. 2010. Use ECP, not ECC, for hard failures in resistive memories. *ACM SIGARCH Computer Architecture News* 38, 3 (2010), 141–152.
- [39] C. H. Sequin and R. D. Clay. 1990. Fault tolerance in artificial neural networks. In *1990 IJCNN International Joint Conference on Neural Networks*.
- [40] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. 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.
- [41] L. Song, X. Qian, H. Li, and Y. Chen. 2017. PipeLayer: A Pipelined ReRAM-Based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 541–552.
- [42] T. Tambe, C. Hooper, L. Pentecost, E. Yang, M. Donato, V. Sanh, A. M. Rush, D. Brooks, and G. Wei. 2020. EdgeBERT: Optimizing On-chip inference for multi-task NLP. *arXiv preprint arXiv:2011.14203* (2020).
- [43] C. Torres-Huitzil and B. Girau. 2017. Fault and error tolerance in neural networks: A review. *IEEE Access* 5 (2017).
- [44] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, and A. Raghunathan. 2017. ScaleDeep: A Scalable compute architecture for learning and evaluating deep networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. 13–26.
- [45] M. Wang, T. Xiao, J. Li, J. Zhang, C. Hong, and Z. Zhang. 2014. Minerva : A scalable and highly efficient training platform for deep learning.
- [46] H. P. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. T. Chen, and M. Tsai. 2012. Metal-oxide RRAM. *Proc. IEEE* 100, 6 (2012), 1951–1970.
- [47] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang. 2017. Stuck-at fault tolerance in RRAM computing systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8, 1 (2017), 102–115.
- [48] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang. 2017. Fault-tolerant Training with On-line fault detection for RRAM-based neural computing systems. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.
- [49] Y. Huai, M. Pakala, Z. Diao, and Y. Ding. 2005. Spin-transfer switching current distribution and reduction in magnetic tunneling junction-based structures. *IEEE Transactions on Magnetics* 41, 10 (2005), 2621–2626.

- [50] F. Yao, A. S. Rakin, and D. Fan. 2020. DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. (2020), 1463–1480.
- [51] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez. 2011. FREE-p: Protecting non-volatile memory against both hard and soft errors. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 466–477.
- [52] J. J. Zhang, T. Gu, K. Basu, and S. Garg. 2018. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. (2018), 1–6.