DEAR: Dependable 3D Architecture for Robust DNN Training

Ashish Reddy Bommana[†], Farshad Firouzi[†], Chukwufumnanya Ogbogu[‡], Biresh Kumar Joardar[§], Janardhan Rao Doppa[‡], Partha Pratim Pande[‡], and Krishnendu Chakrabarty[†]
[†]School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, USA
[‡]School of EECS Washington State University, Pullman, USA
[§]Department of Electrical and Computer Engineering, University of Houston, Houston, USA

***Abstract*—ReRAM-based compute-in-memory (CiM) architectures present an attractive design choice for accelerating deep neural network (DNN) training. However, these architectures are susceptible to stuck-at faults (SAFs) in ReRAM cells, which arise from manufacturing defects and cell wearout over time, particularly due to the continuous weight updates during DNN training. These faults significantly degrade accuracy and compromise dependability. To address this issue, we propose DEAR: dependable 3D architecture for robust DNN training. DEAR introduces a novel online compensation method that employs a digital compensation unit to correct SAF-induced errors dynamically during both forward and backward propagation. This approach mitigates errors induced by SAFs during both the forward and backward phases of DNN training. Additionally, DEAR leverages an HBM-based 3D memory structure to store fault-related error information efficiently. Experimental results show that DEAR limits inferencing accuracy loss to under 2% even when up to 10% of cells are faulty with uniformly distributed faults, and under 2% for up to 5% faulty cells in clustered distributions. This high fault tolerance is achieved with an area overhead of 11.5% and energy overhead of less than 6% for VGG networks and less than 12% for ResNet networks.**

I. Introduction

Deep Neural Networks (DNNs) have revolutionized various domains, including computer vision and natural language processing [1]–[3], but their computational demands—particularly due to numerous matrix-vector multiplication (MVM) operations—create a memory wall bottleneck in traditional Von Neumann-based architectures [4]. To address this challenge, resistive random-access memory (ReRAM)-based compute-in-memory (CiM) architectures offer a promising solution by integrating computation directly into memory, thereby enabling *in situ* execution of MVM operations [5]. Leveraging these capabilities, CiM architectures have demonstrated high-throughput and energy-efficient DNN training [6], [7].

Despite these advantages, DNN training on ReRAM-based CiM architectures must address the reliability challenges inherent to ReRAM cells. These challenges include stuck-at faults (SAFs), where a cell resistance value is fixed and cannot be changed. Such faults typically arise from two primary reasons: manufacturing defects and limited write endurance. These faults can severely impede DNN training, leading to substantial accuracy degradation [8], [9], and ultimately compromising system dependability.

Prior work on SAF tolerance in CiM architectures has predominantly focused on inferencing tasks [10]–[20]. However, these methods are not suitable for DNN training due to frequent weight updates, which require them to be constantly recomputed with each weight update. This severely hinders the training process and adversely impacts throughput. In contrast to inference tasks, there is a limited body of research on training, which can be broadly categorized into two main approaches: (1) remapping [8], [21] and (2) fault-aware training [9], [12], [15], [22]. Remapping techniques aim to dynamically reassign weights between different crossbars based on fault information and the varying fault tolerance levels across different training phases, such as forward and backpropagation [21]. In contrast, fault-aware training methods incorporate fault information into the training process, making the DNN more resilient to SAFs by selectively avoiding updates to weights mapped to faulty cells.

Previously proposed SAF tolerance techniques suffer from several key limitations: (1) *Inaccurate weight updates*: Remapping techniques [8], [21] do not address the faults in the crossbars, as they only involve exchanging weights. Consequently, during the weight update phase, these faults can lead to incorrect weight updates, resulting in accuracy degradation. (2) *Communication overhead:* remapping techniques [21] introduce significant communication overhead, as they require constant coordination and data movement between tiles. This disruption is particularly problematic in crossbar architectures, which rely on maintaining a sequential flow—where the output of one layer directly feeds into the next—is essential [23]. As a result, maintaining efficient inter-layer connections becomes challenging, leading to inefficiencies in the training pipeline. (3) *Low fault coverage*: While fault-aware training can be effective, its success is generally limited to scenarios where the percentage of faults is low, typically around 1-2% [12]. (4) *Stuck-at intermediate levels (SALs)*: All prior works have ignored the presence of SALs, where a cell is stuck at neither fully on (1) nor off (0) [24], [25], which increases the complexity of the algorithms when they are taken into account.

To address the limitations of existing SAF tolerance techniques, we propose DEAR: a dependable 3D architecture for robust DNN training, DEAR introduces a novel online

978-3-9826741-0-0/DATE25/© 2025 EDAA

compensation method to correct errors induced by SAFs. This method dynamically calculates and applies necessary corrections during runtime, eliminating the need for weight remapping. To mitigate the impact of SAFs during weight updates, the digital compensation unit includes a weight correction module to ensure accurate updates. For this method to function effectively, error information associated with faults must be stored and accessed in real-time. Storing such information requires additional memory, which can substantially increase the die footprint in conventional 2D architectures. To overcome this challenge, DEAR employs a heterogeneous 3D architecture based on high-bandwidth memory (HBM), where error logs are stored in a dedicated DRAM tier without increasing the die area. Meanwhile, MVM operations and the compensation process continue in the CiM tier (logic tier).

The key contributions of this paper are as follows:

1) We introduce a novel online compensation method that enhances the robustness of DNN training in CiM by addressing the challenges posed by SAFs.
2) We propose an HBM-based 3D architecture to store error logs associated with SAFs, including SALs, within a dedicated DRAM tier.
3) We establish a theoretical framework that quantifies the fault tolerance limits of DEAR, balancing trade-offs between dependability and system performance within a given overhead limit.

The rest of the paper is organized as follows: Section II provides the background for this work. Section III details the proposed design and introduces the theoretical framework. Section IV outlines the experimental setup and presents simulation results. Finally, Section V concludes the paper.

II. Background

A. Compute-in-memory architecture

Fig. 1 illustrates a typical ReRAM-based CiM architecture, which includes digital-to-analog converters (DACs), crossbar arrays, and analog-to-digital converters (ADCs) integrated within processing elements (PEs). Multiple PE units, along with digital peripherals like activation units, pooling units, and input/output registers, collectively form a tile (T). The crossbar arrays perform analog MVM operations, with the ADCs converting the analog signals into digital values.

The practical limitations of ReRAM crossbars restrict the full activation of all rows and columns within a single clock cycle. These limitations arise primarily due to IR drop and thermal crosstalk, which can negatively impact the accumulated output currents along bitlines [26]–[28]. As a result, only a small part of the crossbar, referred to as the operating unit (OU), can be activated at any given time.

B. Stuck-at-faults

SAFs are permanent defects where the resistance state of a ReRAM cell is fixed at a specific value, rendering the cell unprogrammable. In multi-level ReRAM cells, which can store more than two bits per cell, different types of SAFs are possible corresponding to the multiple levels of conductance states [24], [25]. For a 2-bit cell, the potential SAFs are denoted as SA00, SA01, SA10, and SA11, corresponding to the 2-bit data stored in the cell (00, 01, 10, and 11), where SA01 and SA10 are considered as SALs.

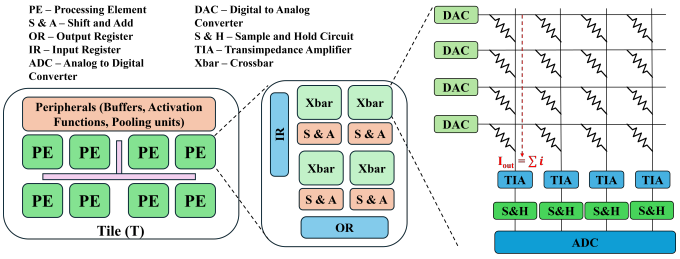


Fig. 1: CiM architecture, which includes a tile (T), processing elements (PEs), crossbars, and other peripheral circuits.

C. DNN training on CiM

DNN training consists of four primary phases: (1) forward pass (FW), (2) activation gradient computation (AG), (3) weight gradient computation (WG), and (4) weight update (WU) phase. In this work, we follow the training methodology outlined in [6], [7], where the FW, AG, and WG phases are pipelined, a process commonly referred to as inter-layer pipelining. Due to the repeated writes involved in the WG phase, MVM computations are performed using SRAM-based CiM [29]. Consequently, the proposed compensation technique is applied only to the FW, AG, and WU phases.

The delay of the pipeline stage, t_{stage}, is determined by the longest delay encountered in any of the FW, AG, or WG phases. It can be expressed as

$$t_{\text{stage}} = \max\{t_{\text{FW}_{\text{max}}}, t_{\text{AG}_{\text{max}}}, t_{\text{WG}_{\text{max}}}\}$$

where $t_{\text{FW}_{\text{max}}}$, $t_{\text{AG}_{\text{max}}}$, and $t_{\text{WG}_{\text{max}}}$ represent the maximum delays in the FW, AG, and WG phases, respectively.

D. HBM-based 3D Architecture

Heterogeneous 3D (H3D) integration overcomes the limitations of traditional 2D designs by improving latency, reducing area footprint, and lowering silicon costs [30], [31]. We adopt H3D integration to stack the DRAM tier on top of the CiM tier. The DRAM tier comprises multiple memory tiles to store the error logs corresponding to the SAF, each corresponding to a ReRAM tile in the CiM tier. Each ReRAM tile is paired with a dedicated memory tile in the DRAM tier. TSVs play a pivotal role in H3D integration by providing vertical connections between stacked dies [32]. TSVs are connected to each memory and ReRAM tile through routers and TSV pads. Fig. 2 depicts the 3D architecture adopted in this work.

III. Proposed online compensation technique

This section presents the hardware design of the proposed digital compensation unit, which includes two key modules: error compensation (EC) and weight correction (WC), each targeting specific phases of the training process. The EC

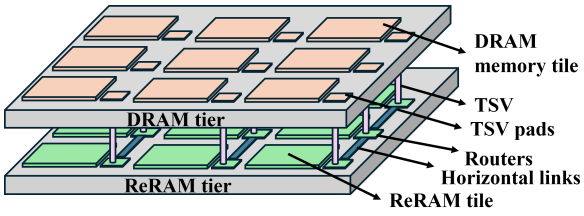


Fig. 2: Proposed DEAR architecture.

module addresses errors during the FW and AG phases, where such errors can propagate and accumulate, potentially compromising the model's accuracy. In contrast, the WC module is dedicated to the WU phase, ensuring accurate weight updates even in the presence of SAFs. Together, these modules provide comprehensive error compensation throughout the training process. First, the necessary mathematical formulations for the EC and WC modules are provided, which are crucial for understanding how to compensate for errors introduced by SAFs. Based on these formulations, the hardware design for both modules is presented. A theoretical framework is introduced to assess the potential compensation achievable with the proposed technique, considering specific hardware configurations and performance constraints.

A. Analysis of fault tolerance

Since the ADC processes bitline currents column-by-column, the errors caused by faults must also be compensated column-by-column at the ADC output. Therefore, we first develop the mathematical formulation that describes how the EC unit computes the compensation for a single column. It is important to note that the faults leading to these errors are distributed across the entire crossbar array, not just confined to individual columns. In Section III-B, we explain how these errors are handled and how the column-by-column compensation process is implemented in hardware.

1) Formulation of Error Compensation: Initially, we consider a single column within an OU of a ReRAM crossbar array that is affected by SAFs. The OU has N rows, with digital 1-bit inputs denoted by $I_1, I_2, \ldots, I_N$ and their corresponding input voltages generated from 1-bit DACs represented as $V_1, V_2, \ldots, V_N$. The relationship between these inputs and voltages is given by:

$$V_i = I_i \times V_{dac}, \text{ for } i = 1, 2, \ldots, N \tag{1}$$

where V_{dac} is the DAC output voltage. Let $w_1, w_2, \ldots, w_N$ denote the weights mapped to the selected column, with corresponding conductance values $G_1, G_2, \ldots, G_N$. Without loss of generality, we assume an equal separation ΔG between neighboring conductance states:

$$\Delta G = \frac{G_{max} - G_{min}}{2^B - 1} \quad (B = 2 \text{ for a 2-bit cell})$$
$$G_i = w_i \times \Delta G + G_{min} \tag{2}$$

The output current I_{out} accumulated along the bitline is converted to an output voltage V_{out} by a TIA (see Fig. 1 with feedback resistance R_f, leading to: $I_{out} = \sum_{i=1}^{N} V_i \cdot G_i$, followed by $V_{out} = I_{out} \cdot R_f$. This voltage is then digitized by an ADC with step size ϵ and resolution $\mathcal{Q}$:

$$V_{max} = N \cdot G_{max} \cdot V_{dac} \cdot R_f, \ \epsilon = \frac{V_{max}}{2^{\mathcal{Q}} - 1} \tag{3}$$

where V_{max} represents the maximum output voltage, occurring when all cells along the column of OU are mapped to G_{max} and all the inputs to the OU's rows are ones. The digital output from the ADC, O_{adc}, is:

$$O_{adc} = V_{out}/\epsilon = \frac{\left(\sum_{i=1}^{N} V_i \cdot G_i\right) \cdot R_f \cdot (2^{\mathcal{Q}} - 1)}{V_{max}} \tag{4}$$

In the presence of SAFs, suppose the conductance of the cells changes to $G'_1, G'_2, \ldots, G'_N$, with the corresponding faulty weights denoted as $w'_1, w'_2, \ldots, w'_N$. Consequently, the change in the conductance value for each cell, ge_i, is given by: $ge_i = G_i - G'_i = (w_i - w'_i) \times \Delta G$ for $i = 1, 2, \ldots, N$

$$ge_i = e_i \times \Delta G \tag{5}$$

where e_i denotes the error in bits in the weight mapped to cell i. The output compensation OC required to correct the faulty output O'_{adc} is given by:

$$O'_{adc} = \frac{\left(\sum_{i=1}^{N} V_i \cdot G'_i\right) \cdot R_f}{\epsilon}$$
$$OC = O_{adc} - O'_{adc} = \frac{\left(\sum_{i=1}^{N} V_i \cdot (G_i - G'_i)\right) \cdot R_f}{\epsilon} \tag{6}$$

Substituting Equations (1), (3) and (5) into Equation (6) yields:

$$OC = \left(\sum_{i=1}^{N} I_i \cdot e_i\right) \cdot \frac{(2^{\mathcal{Q}} - 1) \cdot \Delta G}{N \cdot G_{max}} = K_1 \cdot K_2 \tag{7}$$

In Equation (7), the coefficient K_1 can be determined using digital circuits, such as an adder tree and multiplexers. The coefficient K_2 is fixed for a specific hardware configuration, allowing for the use of a lookup table (LUT) to quickly determine the compensation value corresponding to each K_1. This approach can be applied iteratively to calculate the necessary compensation values for each column in the OU.

2) Formulation of Weight Correction: Similar to the EC module, we present the mathematical foundation for the WC module. Consider a single weight w'_t read from the crossbar in the presence of SAFs, with the ideal weight denoted as w_t, expressed as:

$$w'_t = \sum_{j=0}^{D-1} h'_j \cdot cp^j, \quad w_t = \sum_{j=0}^{D-1} (h'_j + e_j) \cdot cp^j = \sum_{j=0}^{D-1} h_j \cdot cp^j$$

Here, h'_j represents the actual bits of w'_t mapped to the j^{th} cell, e_j denotes the error for that cell, and cp denotes the cell precision, and D represents the number of devices required to map a single weight. Let g_t and α represent the corresponding gradient and learning rate, respectively.

During weight update, the gradient is added to the old weight, resulting in the updated weight w_{t+1}:

$$w_{t+1} = \sum_{j=0}^{D-1} h_j \cdot cp^j + \alpha \cdot g = \sum_{j=0}^{D-1} h''_j \cdot cp^j \tag{8}$$

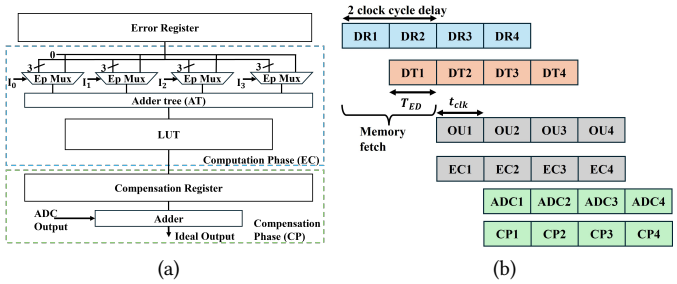
where h''_j represents the bits of the updated weight w_{t+1} in the j^{th} cell. Since the error e_j becomes outdated after the weight update, it must also be updated to maintain accuracy in the next forward phase. The error update is computed indirectly as follows:

$$e_{t+1} = \sum_{j=0}^{D-1} (h''_j - h_j + e_j) \cdot cp^j \tag{9}$$

Here, $h''_j - h_j$ represents the change in the weight in the j^{th} cell. Once the error is updated, subsequent forward phase computations will be error-free.

B. Hardware design for the proposed technique

1) Hardware design of EC: The EC module comprises error registers to store the error information, an adder tree, a LUT, and a single adder, as depicted in Fig. 3(a). The compensation process follows these steps: i) **Memory fetch**: The errors corresponding to an OU are fetched from the DRAM tier and loaded into the error registers within the EC module. This process takes two ReRAM clock cycles, denoted as t_{clk}. Since DRAM typically operates at a higher frequency than ReRAM [33], the data read step can be completed within a single ReRAM clock cycle, even though it requires multiple DRAM clock cycles [5]. The first cycle is used to read the data from DRAM (denoted as DR), and the second cycle is used to transmit the data to the ReRAM tier via TSVs (denoted as DT), as shown in Fig. 3(b). The errors in an OU are stored and loaded based on their index within the OU. For example, in an 8x8 OU, identifying a faulty cell requires 3 bits for the row and 3 bits for the column. The error itself is represented by 3 bits: 2 bits for the error and 1 sign bit, resulting in 9 bits per faulty cell. Once fetched, the errors are loaded into registers by their addresses, and the necessary compensation for each column is computed within the EC module. ii) **Compensation computation phase (EC)**: The EC module sequentially calculates the compensation required for each column. For a single column, the dot product between the input I_i and the corresponding errors are computed using multiplexers (Ep Mux) that selectively propagate or mask errors based on I_i as shown in Fig. 3(a). Next, an adder tree aggregates these weighted errors to compute K_1 as outlined in Equation (7). The final compensation value, $K_1 \cdot K_2$ (Equation (7)), is determined by using the LUT within the EC module. This process is repeated iteratively for each column in the OU, with the computed compensation values stored in compensation registers (see Fig. 3(a)). iii) **Compensation phase (CP)**: The compensation values stored in the compensation registers are sequentially added to the ADC output using an adder as shown in Fig. 3(a), by processing each column individually [5]. This compensation phase is performed concurrently with the ADC, as illustrated in Fig. 3(b). To ensure error compensation across the entire crossbar, this process is repeated for each OU in a pipelined manner, as illustrated in Fig. 3(b). In each clock cycle, one OU operation is performed while the EC module concurrently completes the calculation of the compensation required for that OU.



(a) (b)

Fig. 3: (a) Illustration of EC module for a 4x4 OU size; (b) Pipeline of OU when integrated with the EC module.

2) Estimating data transfer time: The total amount of error data (D_{ED}) to be fetched from DRAM depends on the bit width of the error (b_e), the number of rows and columns in an OU (MxN), the fault tolerance level in an OU, denoted by α, where $\alpha \in [0, 1]$ represents the ratio of protected cells to the total number of cells in the OU, and the number of crossbars in a tile (n_{xb}). This can be expressed as: $D_{ED} = (b_e \cdot M \cdot N \cdot \alpha \cdot n_{xb})$. The total time required to transfer D_{ED}, denoted by T_{ED} (see DT cycle in Fig. 3(b)), depends on the number of TSVs per tile (n_{tsv}) and the bit rate of TSV (br_{tsv}). This is given by:

$$T_{ED}\ (\text{ns}) = \frac{D_{ED}}{br_{tsv} \cdot n_{tsv}} = \frac{b_e \cdot M \cdot N \cdot \alpha \cdot n_{xb}}{br_{tsv} \cdot n_{tsv}} \tag{10}$$

As observed in Fig. 3(b), the error logs corresponding to SAFs in each OU must be fetched before the OU is activated to ensure the throughput of the training remains unaffected. However, in case T_{ED} exceeds the ReRAM clock period (t_{clk}), the clock period must be extended to allow sufficient time for data to be loaded from DRAM to the registers before OU activation. This extension can result in performance overhead, depending on the volume of data being transferred, potentially limiting the amount of compensation that can be applied without adversely impacting performance.

3) Hardware design of WC: During the weight update, prior to adding the gradient, the incorrect weight read from the crossbar is corrected using the error, as described in Equation (8). This correction is implemented by reusing the adder from the EC module. Subsequently, the error is updated based on the updated weight. The difference $h''_j - h_j$ in Equation (9) is inherently computed during the weight update process [29]. Thus, no additional calculation is required. The only additional hardware needed is an extra adder for updating the error, resulting in minimal overhead for the WC module—essentially just one additional adder.

C. Theoretical characterization of fault tolerance

As discussed in Section III-B2, there is a limit to how much compensation can be applied in an OU before performance starts to degrade. To formalize the limits of the proposed compensation technique, we present a theorem that determines the maximum fault tolerance achieved in an OU, given the predefined hardware configuration of the CiM architecture and the constraints on performance overhead. Such analysis is necessary to provide a theoretical characterization and quantification of the dependability offered by DEAR. The theorem is stated as follows:

Theorem 1. *An upper bound on the fault tolerance level α achieved in an OU without exceeding the performance overhead limit p is given by:*

$$\alpha \leq (p+1) \cdot \frac{t_{stage}}{Z} \cdot \frac{t_{clk} \cdot br_{tsv} \cdot n_{tsv}}{b_e \cdot M \cdot N \cdot n_{xb}}, \quad (11)$$

$$\text{where } Z = \begin{cases} t_{FW_{max}} \text{ if } t_{FW_{max}} > t_{AG_{max}}, \\ t_{AG_{max}} \text{ otherwise.} \end{cases} \quad (12)$$

Proof. Consider the scenario where $T_{ED} > t_{clk}$ and has an fault tolerance level of α . In this case, the new clock period t'_{clk} will be equal to T_{ED}. Consequently, the maximum latencies of the FW phase ($t_{\text{FW}_{\text{max}}}$) and AG phase ($t_{\text{AG}_{\text{max}}}$) will also increase. Let these new latencies be denoted as $t'_{\text{FW}_{\text{max}}}$ and $t'_{\text{AG}_{\text{max}}}$, respectively. The new pipeline stage delay t'_{stage} is therefore given by $t'_{stage} = \max\{t'_{\text{FW}_{\text{max}}}, t'_{\text{AG}_{\text{max}}}, t_{stage}\}$. In the worst-case scenario where there is an overhead, the new pipeline state delay t'_{stage} is $\max\{t'_{\text{FW}_{\text{max}}}, t'_{\text{AG}_{\text{max}}}\}$. To quantify these new latencies, we first define the following:

$$Nc_{FW_{max}} = \frac{t_{\text{FW}_{\text{max}}}}{t_{clk}}, \quad Nc'_{FW_{max}} = Nc_{FW_{max}} + 2$$

Here, $Nc_{FW_{max}}$ and $Nc'_{FW_{max}}$ represent the number of clock cycles required to complete the forward pass without and with the proposed technique, respectively. The additional two cycles account for data reading and transfer delays (see Fig. 3(b)). Therefore, $t'_{\text{FW}_{\text{max}}}$ is given by:

$$\begin{aligned} t'_{\text{FW}_{\text{max}}} &= Nc'_{FW_{max}} \cdot t'_{clk} \\ &= \left(\frac{t_{\text{FW}_{\text{max}}}}{t_{clk}} + 2\right) \cdot \frac{b_e \cdot M \cdot N \cdot \alpha \cdot n_{xb}}{br_{tsv} \cdot n_{tsv}} \\ &\approx t_{\text{FW}_{\text{max}}} \cdot l, \text{ where } l = \frac{b_e \cdot M \cdot N \cdot \alpha \cdot n_{xb}}{br_{tsv} \cdot n_{tsv} \cdot t_{clk}} \end{aligned} \quad (13)$$

Case 1: If $t_{\text{FW}_{\text{max}}} > t_{\text{AG}_{\text{max}}}$, then $t'_{stage} = t'_{\text{FW}_{\text{max}}}$. To ensure that the performance overhead remains within the allowable limit p, the following inequality must hold:

$$\begin{aligned} \frac{t'_{stage} - t_{stage}}{t_{stage}} \leq p &\iff t'_{\text{FW}_{\text{max}}} \leq (p+1) \cdot t_{stage} \\ &\implies l \leq (p+1) \cdot \frac{t_{stage}}{t_{\text{FW}_{\text{max}}}} \end{aligned} \quad (14)$$

Substituting the expression for l from Equation (13) and solving for α results in:

$$\alpha \leq (p+1) \cdot \frac{t_{stage}}{t_{\text{FW}_{\text{max}}}} \cdot \frac{t_{clk} \cdot br_{tsv} \cdot n_{tsv}}{b_e \cdot M \cdot N \cdot n_{xb}} \quad (15)$$

Parameter	Value
n_{xb}	96
t_{clk} (ns)	10
OU Size (MxN)	8 x 8
n_{tsv}	512
b_e	9
br_{tsv} (no. of bits/ns)	2
p	0.05

TABLE I: Hardware configuration adopted for evaluation of DEAR.

Therefore, the condition stated in the theorem is proven. A similar reasoning can be applied to **Case 2** ($t_{\text{FW}_{\text{max}}} \leq t_{\text{AG}_{\text{max}}}$).

To illustrate the practical significance of Theorem 1, consider the hardware configuration presented in Table I and the ResNet18 model. Using NeuroSim [29] simulation, the ratio $t_{stage}/t_{\text{AG}_{\text{max}}}$ is determined to be 1.015. By applying the bound established in Theorem 1, we derive $\alpha < 0.19$. This implies that if the percentage of faults exceeds 19% in an OU, the proposed method cannot guarantee 100% fault tolerance without incurring a performance overhead. This example demonstrates how the derived bound can be applied to determine the optimum level of fault tolerance achievable while ensuring that the performance overhead remains within predefined limits.

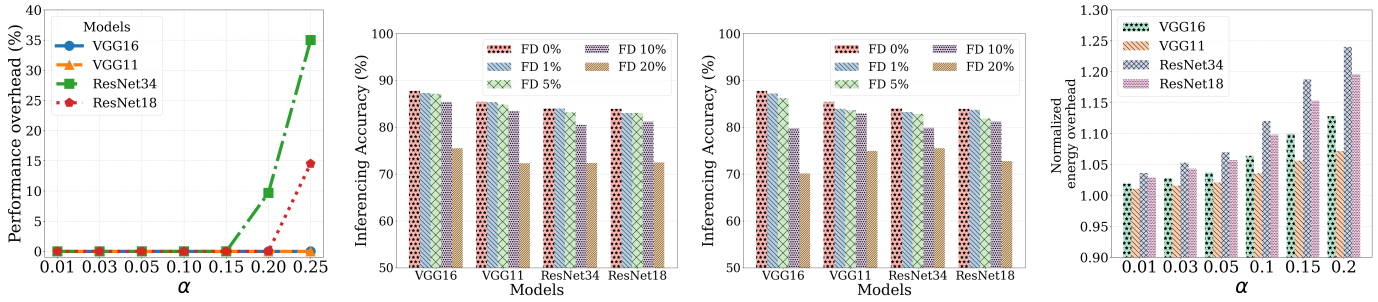
IV. Experiments and Results

A. Experimental Setup:

Simulation Tools: To estimate the performance and energy consumption of each layer in the model, we employed NeuroSim [29]. For evaluating DNN training on CiM architectures in the presence of faults, we utilized PytorX [22]. The digital circuits in the EC and WC modules were synthesized using Synopsys Design Compiler on a nangate 45 nm technology node, with results scaled down to the 32 nm technology. The DRAM read and write energy were modeled through CACTI [34], and the TSV data transfer speed and energy per bit were adopted from [35], [36], respectively. **DNN Models and Dataset:** We evaluated variants of VGG [1] and ResNet [2] models, all trained on the CIFAR-10 dataset. CIFAR-10 consists of 60,000 RGB images, 50,000 for training and 10,000 for testing.

B. Performance overhead

As inferred from Section III-C, there is a limit to the amount of protection that can be achieved with the proposed method. As per Equation (10), T_{ED} is directly proportional to α; as α increases, T_{ED} can exceed t_{clk}, leading to additional delay in processing the layers in the training phase. This delay can introduce performance overhead. For instance, consider a layer l with a delay t_l, which is less than the delay of pipeline stage t_{stage}. When an additional delay t_d is introduced by EC, the new delay for that layer is $t_l + t_d$. If this new delay exceeds t_{stage}, then this results in performance overhead. Conversely, if $t_l + t_d$ remains within t_{stage}, the additional delay is masked within the pipeline, and no overhead occurs. Fig. 4(a) presents the performance overhead for various models. Notably, for ResNet34, there is



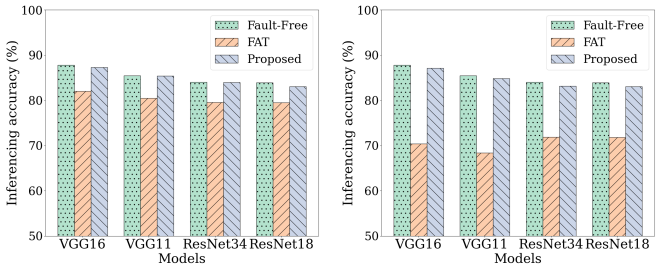
(a) (b) (c) (d)

Fig. 4: (a) Performance overhead of various models as α is varied. (b)-(c) Accuracy trends observed for uniform and clustered fault distributions when α is 0.1, respectively. (d) Normalized energy overhead for various models at different α values.

an overhead of 10% when α is 0.2. These results confirm that the level of protection achievable with the proposed method is subject to the limits imposed by Theorem 1.

C. Accuracy results

We evaluated the proposed method under fault densities, defined as the percentage of cells that are faulty, ranging from 1% to 20%, and analyzed its effectiveness under two types of fault distributions: uniform and clustered. For the clustered distribution, we adopted a Gaussian distribution-based clustering algorithm, as described in [37]. Fig. 4(b) and Fig. 4(c) depict the accuracy trends for various models under these distributions at different fault densities, with α set to 0.1, respectively. Notably, the accuracy degradation remains below 2% for all models, even at a 10% fault density in the uniform case. In contrast, for clustered distributions, we observe a more significant accuracy drop of 3–6%. This is likely due to some OUs experiencing 100% faults, for which the proposed method offers limited protection, as constrained by the bounds derived in Theorem 1, leading to a larger accuracy loss. Fig. 5(a) and Fig. 5(b) compares the proposed method with fault-aware training (FAT) [22] under uniformly distributed faults. The results show that the proposed method consistently outperformed fault-aware training across all scenarios.



(a) (b)

Fig. 5: (a)-(b) Accuracy trends comparing the proposed method with FAT at 1% and 5% fault densities, respectively.

D. Energy and Area overhead

We estimated the energy consumption of the proposed design for various α values, as shown in Fig. 4(d). The total energy for the compensation process includes DRAM read/write energy, data transfer energy through TSVs, and energy used by the EC and WC modules. As illustrated in Fig. 4(d), the energy overhead remains below 6% for VGG variants and below 12% for ResNet variants when α is 0.1. Fig. 6 shows the energy breakdown across different components in DEAR for VGG16 and ResNet18. The proposed design maintains an area overhead of 11.5% at the PE level. This overhead was calculated using the synthesis reports, with the baseline PE described in [5].

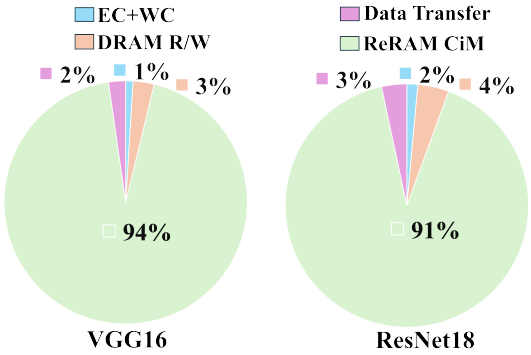


Fig. 6: Energy breakdown for different components in DEAR for VGG16 and ResNet18 models.

V. Conclusion

We introduced DEAR, a dependable HBM-based 3D architecture for robust DNN training on ReRAM-based CiM, incorporating an online compensation method with EC and WC modules for tolerating errors due to SAFs. Our theoretical framework shows DEAR's effective fault tolerance, balancing dependability with minimal performance overhead. Experiment results show that DEAR maintains inferencing accuracy within 2% loss at 10% fault density for uniformly distributed faults and under 5% for clustered faults, while digital compensation unit results in only 11.5% area overhead at the PE level. Energy overhead remains below 6% for VGG and 12% for ResNet models.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, 2017.

[4] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014.

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

[6] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2017.

[7] H. Jiang, S. Huang, X. Peng, and S. Yu, "MINT: Mixed-precision RRAM-based in-memory training architecture," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[8] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training enabled by on-line fault detection for RRAM-based neural computing systems," *IEEE TCAD*, no. 9, 2019.

[9] B. K. Joardar, J. R. Doppa, H. Li, K. Chakrabarty, and P. P. Pande, "Learning to train CNNs on faulty ReRAM-based manycore accelerators," *ACM Transactions on Embedded Computing Systems*, 2021.

[10] F. Zhang and M. Hu, "Defects mitigation in resistive crossbars for analog vector matrix Multiplication," in *Asia and South Pacific Design Automation Conference*, 2020.

[11] H. Shin, M. Kang, and L.-S. Kim, "Fault-free: A framework for analysis and mitigation of stuck-at-fault on realistic ReRAM-based DNN accelerators," *IEEE Transactions on Computers*, no. 7, 2023.

[12] W. Li, Y. Wang, H. Li, and X. Li, "RRAMedy: Protecting ReRAM-based neural network from permanent and soft faults during its lifetime," in *IEEE International Conference on Computer Design*, 2019.

[13] M. Oli-Uz-Zaman, S. A. Khan, W. Oswald, Z. Liao, and J. Wang, "Stuck-at-fault immunity enhancement of memristor-based edge AI systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, no. 4, 2022.

[14] C. Quan, M. E. Fouda, S. Lee, G. Jung, J. Lee, A. E. Eltawil, and F. Kurdahi, "Training-free stuck-at fault mitigation for ReRAM-based deep learning accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[15] Q. Xu, J. Wang, B. Yuan, Q. Sun, S. Chen, B. Yu, Y. Kang, and F. Wu, "Reliability-driven memristive crossbar design in neuromorphic computing systems," *IEEE Transactions on Automation Science and Engineering*, no. 1, 2023.

[16] S. T. Ahmed, R. Rakhmatullin, and M. B. Tahoori, "Online fault-tolerance for memristive neuromorphic fabric based on local approximation," in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–4.

[17] G. Charan, A. Mohanty, X. Du, G. Krishnan, R. V. Joshi, and Y. Cao, "Accurate inference with inaccurate rram devices: A joint algorithm-design solution," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 6, no. 1, pp. 27–35, 2020.

[18] C.-Y. Chen and K. Chakrabarty, "Efficient identification of critical faults in memristor-based inferencing accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 7, pp. 2301–2314, 2022.

[19] A. Eldebiky, G. L. Zhang, G. Böcherer, B. Li, and U. Schlichtmann, "CorrectNet: Robustness enhancement of analog in-memory computing for neural networks by error suppression and compensation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[20] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Handling stuck-at-faults in memristor crossbar arrays using matrix transformations," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 438–443.

[21] C.-H. Tung, B. K. Joardar, P. P. Pande, J. R. Doppa, H. H. Li, and K. Chakrabarty, "Dynamic task remapping for reliable CNN training on ReRAM crossbars," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[22] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," in *Design Automation Conference*, 2019.

[23] S. T. Ahmed, M. Hefenbrock, C. Münch, and M. B. Tahoori, "Neuro-Scrub+: Mitigating retention faults using flexible approximate scrubbing in neuromorphic fabric based on resistive memories," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 5, pp. 1490–1503, 2023.

[24] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu, "Modeling, detection, and diagnosis of faults in multilevel memristor memories," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, no. 5, 2015.

[25] M. Zhao, B. Gao, Y. Xi, F. Xu, H. Wu, and H. Qian, "Endurance and retention degradation of intermediate levels in filamentary analog RRAM," *IEEE Journal of the Electron Devices Society*, 2019.

[26] M.-Y. Lin, H.-Y. Cheng, W.-T. Lin, T.-H. Yang, I.-C. Tseng, C.-L. Yang, H.-W. Hu *et al.*, "DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning," in *IEEE/ACM International Conference on Computer-Aided Design*, 2018.

[27] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue *et al.*, "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *IEEE International Solid-State Circuits Conference*, 2018.

[28] Y. Cai, Z. Wang, Z. Yu, Y. Ling, Q. Chen, Y. Yang, S. Bao *et al.*, "Technology-array-algorithm co-optimization of RRAM for storage and neuromorphic computing: Device non-idealities and thermal cross-talk," in *IEEE International Electron Devices Meeting*, 2020.

[29] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2306–2319, 2021.

[30] B. K. Joardar, A. Deshwal, J. R. Doppa, P. P. Pande, and K. Chakrabarty, "High-throughput training of deep CNNs on ReRAM-based heterogeneous architectures via optimized normalization layers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[31] Z. Wan, C.-K. Liu, M. Ibrahim, H. Yang, S. Spetalnick, T. Krishna, and A. Raychowdhury, "H3DFact: Heterogeneous 3D integrated CIM for factorization with holographic perceptual representations," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024.

[32] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "HBM (High Bandwidth Memory) DRAM technology and architecture," in *IEEE International Memory Workshop (IMW)*, 2017.

[33] S. Yu, *Semiconductor memory devices and circuits.* CRC Press, 2022.

[34] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 3–14.

[35] B. K. Joardar, K. Duraisamy, and P. P. Pande, "High performance collective communication-aware 3D Network-on-Chip architectures," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1351–1356.

[36] J.-Y. Kim, T. Kim, J. You, K. Kim, B. M. Moon, K. Sohn, and S.-O. Jung, "An energy-efficient design of TSV I/O for HBM With a data rate up to 10 Gb/s," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 11, pp. 3242–3252, 2023.

[37] A. Chaudhuri, B. Yan, Y. Chen, and K. Chakrabarty, "Hardware Fault Tolerance for Binary RRAM Crossbars," in *IEEE International Test Conference*, 2019.