Functional Criticality Analysis of Structural Faults in AI Accelerators*

Arjun Chaudhuri†, Jonti Talukdar†, Fei Su‡, and Krishnendu Chakrabarty†
†Department of Electrical and Computer Engineering, Duke University, Durham, NC
‡Intel Corporation, Folsom, CA

***Abstract*—The ubiquitous application of deep neural networks (DNN) has led to a rise in demand for artificial intelligence (AI) accelerators. For example, the Tensor Processing Unit from Google—based on a systolic array—and its variants are of considerable interest for DNN inferencing using AI accelerators. This paper studies the problem of classifying structural faults in such an accelerator based on their functional criticality. We first analyze pin-level faults in the processing elements (PEs) of a systolic array. Simulation results for the LeNet network with 8-bit fixed-point, 16-bit floating-point, and 32-bit floating-point data paths applied to the MNIST dataset show that over 93% of the pin-level structural faults in a PE are functionally benign. We present a greedy iterative framework for determining the criticality of stuck-at faults in a PE netlist and analyze the limitations of criticality analysis methods based on repeated fault simulations. We next present a scalable two-tier machine-learning (ML) based method to assess the functional criticality of stuck-at faults in a computationally efficient manner. We address the problem of minimizing misclassification by utilizing generative adversarial networks (GANs). Two-tier ML/GAN-based criticality assessment leads to less than 1% test escapes during functional criticality evaluation of structural faults.**

I. Introduction

In recent years, deep neural networks (DNNs) have emerged as a workhorse for data-intensive applications such as image classification and segmentation, voice recognition, and natural language processing [1]. A DNN can achieve high performance due to the large number of model parameters (weights) trained using backpropagation algorithms [2], non-linear activations [1], and the use of convolution layers [3].

The ubiquitous application of DNNs has led to a rise in demand for custom artificial intelligence (AI) accelerators [4]. AI hardware accelerators can be classified on the basis of two major use-cases: training and inferencing. Training, being an iterative and compute-intensive process, is typically carried out in the cloud using high-performance computing clusters. On the other hand, inferencing is carried out by using AI accelerators on edge devices [5] as well as in datacenters [6]. Systolic array-based AI accelerators have become increasingly popular for inferencing because of their high throughput, regularity, and simplicity of design [7], [8]. They are being deployed for inferencing in a number of applications, e.g., autonomous driving, manufacturing automation, and navigation [9].

Many such use-cases require high reliability. Built-in self-test (BIST) can be used for enabling power-on self-test in order to detect in-field failures [10]. However, DNN inferencing applications such as image classification are inherently fault-tolerant with respect to structural faults; it has been shown that many faults are not functionally critical, i.e., they do not lead to any significant error in inferencing [11], [12]. As a result, conventional pseudo-random pattern generation for targeting all faults with BIST is an "over-kill".

Prior work to evaluate the fault tolerance of DNNs does not provide any computationally efficient method to assess the functional criticality of structural faults [12], [13]. The *functional criticality* of a fault is determined by the severity of its impact on functional performance [14]. In the context of AI inferencing accelerators, it is determined by the impact of the fault on inferencing accuracy, measured with respect to a given criticality threshold. The functional criticality of a structural fault (such as stuck-at), along with the criticality thresholds, depend on the application, input data, DNN-to-accelerator mapping, and the accelerator architecture.

In this paper, we systematically analyze structural faults in AI accelerators. Our main contributions are as follows:

- Insights into the functional criticality of pin-level faults in the processing elements (PEs) of an AI accelerator based on a systolic array.
- Theoretical insights that enable the quick identification of a significant number of functionally benign faults.
- A greedy iterative framework based on the above theoretical insights for identifying functionally benign faults in the PE netlist.
- A two-tier DNN-based method to classify faults based on functional criticality for a given inferencing application.
- An analysis method based on generative adversarial networks to reduce the number of misclassifications.
- An analysis method to determine the minimum tolerable multi-frame inferencing accuracy of accelerators in autonomous systems.

We evaluate the proposed method by mapping the LeNet DNN [15] to a systolic array and analyzing the functional criticality of structural faults for handwritten digit classification. We choose LeNet as the application dataset to demonstrate the proposed method because it is a good representative of a typical convolutional neural network and consists of fully connected, convolutional, and pooling layers. The DNN architecture, DNN-to-hardware mapping, and application dataset are user-defined inputs to our ML-based criticality evaluation framework. Therefore, the proposed method is general and can be extended to other DNN architectures, mappings, and applications.

*A preliminary version of this paper was published in Proc. IEEE International Test Conference, 2020.

The remainder of the paper is organized as follows. Section II reviews systolic arrays and motivates functional criticality analysis. Section III describes the DNN architecture used in this work and its mapping to the systolic array, along with the fault-injection framework. Section IV presents insights derived from pin-level fault criticality analysis for PEs. Section V presents a greedy iterative method for determining functionally benign faults in the PE netlist. Section VI presents the two-tier DNN-based criticality evaluation method and Section VII presents the experimental results. Finally, Section VIII concludes the paper.

II. Background and Motivation

A. DNN Accelerators based on a Systolic Array

AI accelerators based on a systolic array provide high throughput, high compute density, simplified control, and regular design [6], [4]. A systolic array consists of a two-dimensional array of PEs that perform matrix multiplication through multiply and accumulate (MAC) operations on incoming data streams. Each PE consists of a multiplier and an adder; it interacts with its neighboring PEs such that data streaming down the column of the array is accumulated while the other input is broadcast across all PEs in the same row. This design allows multiple uses of each input data item (across columns) and obviates the need for buffering between PEs.

The PE includes an IEEE Std. 754 floating-point (FP) adder and multiplier unit along with corresponding weight and partial-sum registers (Fig. 1). Each PE multiplies an input activation X_r with weight $W_{r-1,c}$ and adds the product to a previously accumulated sum $S_{r-1,c}$ passed down from the PE immediately preceding it in the same column. The weight register temporarily holds $W_{r-1,c}$ before passing it to the PE immediately following it in the same column. The partial-sum register temporarily holds the newly accumulated sum $S_{r,c}$ before passing it down to the next PE. As an example, Google's Tensor Processing Unit (TPU) utilizes a 256×256 systolic array-based inferencing engine that performs 8-bit integer multiply and 16-bit integer accumulate operations and is 15-30× faster than contemporary CPUs or GPUs [6], [16]. More recent versions of the TPU (v2 and v3) also support 32-bit floating-point (FP) operations with bfloat-16 multiplication [17].

B. Functional Criticality Analysis of Structural Faults

Functional Fault Criticality: The application of DNN algorithms to specific use-cases (classification, regression, segmentation, etc.) provides some flexibility in terms of the acceptable tolerance required to ensure functional correctness. Examples include accuracy thresholding for classification [18] and the residual error in regression [19]. This flexibility paves the way for additional metrics to evaluate and grade structural faults based on functional criticality, beyond traditional methods such as fault coverage and structural test quality.

A structural fault is considered to be *functionally critical* if it leads to functional failure. For DNN accelerators, functional failure can be evaluated in terms of the fault's impact on

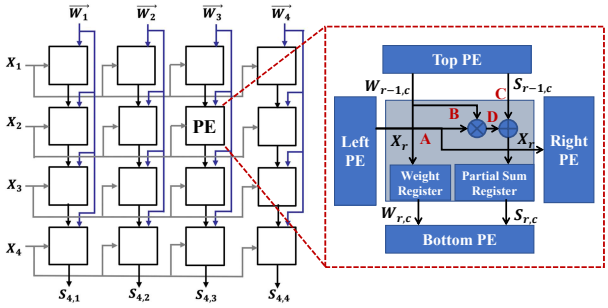

Fig. 1: A systolic array and the architecture of a PE.

inferencing accuracy. A fault is deemed to be *benign* if it does not affect the inferencing accuracy for the use-case network. The accuracy threshold used for classifying faults as being benign or critical can be pre-determined based on the accuracy requirement and safety criticality of the use-case application.

In addition, testing efforts for AI accelerators can benefit from the inherent resiliency of machine learning, especially for convolutional neural networks (CNNs) [12]. Prior work [11] suggests that CNNs can tolerate random variations and noise in their inputs; this level of tolerance can be attributed to the robust training process involving the updating of weights and biases through gradient descent over large datasets. In addition, regularization features such as dropout, normalization, and loss minimization [20] make CNNs robust to certain types of structural faults and soft errors [12].

The functional criticality of structural faults also depends on additional factors that go beyond the robustness of training. These factors include the location of the fault site, the dataflow through the array, the mapping of the DNN model to the underlying systolic array hardware, distribution of the weights and biases, as well as the diversity of the input data. Recent work [21] investigates functional testing for a similar DNN architecture (LeNet), but it only samples randomly selected structural faults and overlooks hardware reuse, which is typically how DNNs are mapped to the AI accelerator. Other recent works on systolic array accelerator architectures limit the analysis to functional errors that result from faults injected only at the PE level [13], [22], [23]. Gate-level fault simulation is necessary to identify functionally critical faults in order to reduce the number of seed patterns for deterministic BIST and quantify the robustness of the DNN-to-hardware mapping. The PE-level fault or error injection does not achieve the granularity of fault simulation required for reducing pattern count by exploiting the inherent fault tolerance of AI accelerators.

Importance of Functional Criticality Assessment: DNN-specific functional criticality analysis identifies faults that cause measurable and significant deviations from acceptable requirements. The criticality of faults can thus be used to guide test grading and quality assessment of AI accelerators at various stages of the product lifecycle. Fault-criticality assessment method can be used to target domain-specific use-cases—different models, e.g., AlexNet, GoogleNet, ResNet, etc., mapped to the accelerator for applications such as digit classification and pattern recognition. As the functional-criticality of faults depends on the type of dataset, DNN

model, and model-to-hardware mapping, individual catalogs containing lists of critical and benign faults can be pre-generated for every domain-specific use-case. These catalogs can then be used by the customer as a reference for assessing functional fault-criticality. Potential applications include guiding selective test-point insertion for critical faults so that they are always testable during manufacturing testing. The design-for-test effort for test-point insertion can be reduced by appropriately deemphasizing benign faults. Moreover, pattern generation for BIST and in-field testing can be targeted at functionally critical faults across all the catalogs. Benefits include reduced pattern count and test time during in-field test. Finally, functional criticality analysis can also guide selective fault-mitigation techniques for critical fault sites.

Error Propagation: A single fault in a PE can result in multiple erroneous neurons. Due to array reuse across multiple passes to compute the final result, the data stream is contaminated after the first pass of the network through the systolic array, thereby corrupting the data stream throughout the array for all subsequent passes. The resulting error contamination, along with the non-linearity of each DNN layer, makes it especially difficult to track the propagation of erroneous values.

One approach to analyzing functional criticality of faults is to carry out fault simulation using the PEs. Fault injection in a PE can be used to accumulate errors after each pass of the systolic array. This iterative approach is analogous to performing functional fault simulation of the neural network mapped to the systolic array, as the fault is present during the computation of each layer.

However, the above method is impractical because of the prohibitively high CPU time. A single 32-bit PE can easily contain tens of thousands of stuck-at faults. While the actual number of faults depends on the PE architecture, the total number of structural faults for a 256×256 PE array, as in the Google TPU [6], is in the range of billions. Even with advanced computing resources and extensive parallelization, it will take years to classify all these faults using brute-force functional simulation on a given input dataset.

III. DNN-to-Hardware Mapping and Fault-Injection Framework

A. LeNet-5M Architecture

LeNet-5 is a deep CNN architecture originally designed for handwritten digit recognition [15]. We use a slightly modified version called LeNet-5M, which consists of two convolutional layers, three fully connected layers, and two subsampling layers; all convolutional and fully connected layers are followed by rectified-linear units (ReLU). The subsampling layers in LeNet-5M perform average pooling as compared to pooling with a trainable parameter in the original LeNet-5. Average pooling facilitates hardware reuse because it can be formulated as a special case of 3D-convolution, thereby obviating the need for specialized logic to perform max-pooling operation.

The input to the network is a 28×28 grayscale image with zero padding, creating a 32×32 input frame. The first convolutional layer (C1) contains six filters (kernels) of size

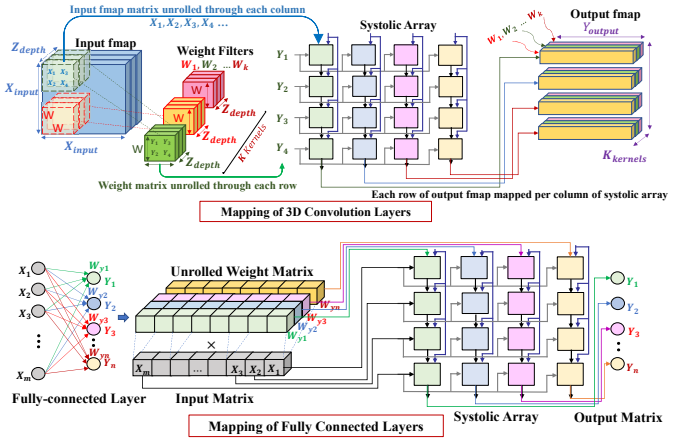

Fig. 2: Mapping strategy used in this work for both 3D convolution layers and fully connected layers.

5×5. The second convolutional layer (C2) contains 16 filters of size 5×5. The three fully connected layers (including the output layer) contain 120, 84, and 10 neurons, respectively. The 10 neurons in the output layer represent the confidence values for the 10 individual classes in the dataset.

This network was designed and trained in MATLAB on a dataset of 7,500 images from the MNIST benchmark [15] with an equal number of images in all 10 categories (digits "0" to "9"). The input data includes 100 images with 10 images from each class. This data is used to evaluate and characterize the impact of different structural faults in the hardware with respect to the baseline model performance.

B. Hardware Mapping

We designed and implemented a 128×128 systolic array for the AI accelerator with both the 32-bit and 16-bit IEEE FP data formats. All design configurations were first simulated at the PE level using Python followed by RTL design and logic synthesis using Verilog HDL.

The overall mapping strategy is illustrated in Fig. 2. Note that the mapping is an user-defined input to the proposed criticality evaluation framework. In this work, we have adopted a layer-specific optimal mapping strategy for each layer type. For convolutional layers, the 3D input feature maps and weights were mapped in such a way that the weights were fixed at the rows of the systolic array while the inputs were rolled in through each column. This is similar to weight-stationary mapping [4]. Both weights and inputs were unrolled in column-major format, allowing twice the throughput compared to unrolling in the row-major format. Convolution was done one output feature slice at a time. Each column produces one row of the output feature map.

Mapping for average pooling layers was done as a special case of 3D convolution, where convolution was mimicked with weight kernels of size $2 \times 2 \times Z \times K$ with each cell having a value of 0.25, Z being the number of input channels and K being the number of weight kernels. Mapping for fully connected layers was done in a fashion similar to 3D convolution layers, with the key difference being that the

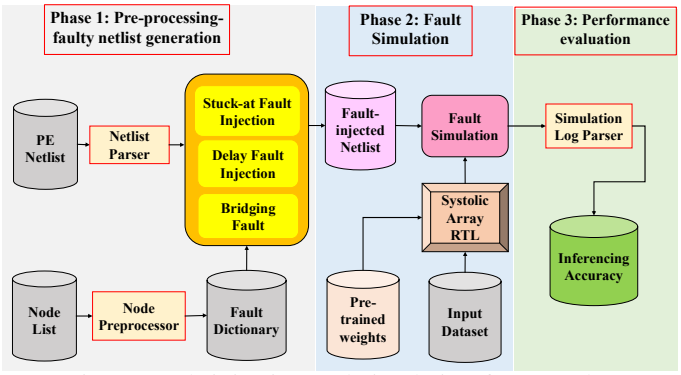

Fig. 3: Fault injection and simulation framework.

weights and inputs were reversed with each other. This is similar to the input-stationary mapping [4]. Bias addition was done using a separate set of adders outside the systolic array. Note that different mapping strategies lead to error propagation through the network in different ways, leading to potentially different functional impacts for the same structural fault.

C. Fault-Injection Framework

We have implemented a framework (Fig. 3) to inject structural faults within both behavioral and structural models in Python and Verilog HDL, respectively, of the DNN mapped to the systolic array. In this way, we can study the impact of injected faults on the classification performance of the DNN, thereby allowing the criticality assessment of each fault.

The fault-injection framework supports stuck-at-faults (s-a-0, s-a-1), bridging faults to model shorts through wired-AND wired-OR approaches, and delay faults to model opens. While the framework supports all the above fault models, the analysis and results reported in this paper are limited to single-stuck-at faults. Fault injection is done inside the PE, i.e., the adder and multiplier macros as well as its periphery at the pin-level. The impact of the injected fault on the inferencing accuracy is assessed using the input dataset.

IV. Pin-Level Fault Criticality Analysis

A. Evaluation of Pin-Level Fault Criticality

A 128×128 systolic array was designed in Python and automated pin-level fault injection carried out as explained in Section III-C. As shown in Fig. 1, a PE in the systolic array consists of three input buses, namely A, B, and C, where A and B are inputs to the multiplier, while C is the accumulation input bus to the adder within the PE. The multiplier output-bus internal to the PE is denoted as bus D. As each bus is 32 bits wide, a total of 256 faults are injected and simulated for functional impact in case of a 32-bit PE. Similarly, 128 faults are injected for a 16-bit PE. Each pin of the bus is functionally simulated for both s-a-1 and s-a-0 faults.

The pin-level fault-injection experiments were performed for both 16-bit and 32-bit PE architectures. Results on pin-level structural fault criticality are shown in Fig. 4. The fault-free inferencing accuracies are 98% and 99% for 16-bit and 32-bit architectures, respectively. We only present results for a 16-bit PE in row 1 column 0, i.e., PE(1,0), and 32-bit PEs in rows 1 and 75 of column 0, i.e., PE(1,0) and PE(75,0), due to lack of space. We have simulated 185 PEs (85 32-bit PEs and 100 16-bit PEs) across multiple columns and observed similar results in each case; more results are presented in [24]. For our use-case and dataset, a 95% classification accuracy exceeds what is accepted in practice [15]. We set an exceptionally high accuracy threshold of 95% for the 32-bit PE architecture, higher than for typical AI applications [25], because of two reasons: (1) ensure that fault-criticality assessment is rigorous, (2) generate an adequate number of critical faults for training machine learning models (as described later) to analyze PE-internal faults. Thus, a fault in the 32-bit PE was deemed to be functionally critical if the classification accuracy was less than 95%. Similarly, we set the criticality threshold to 90% for the 16-bit architecture. We note that more than 90% of the faults are functionally benign. Moreover, the 32-bit and 16-bit PE architectures show nearly identical fault-criticality behavior. This finding can be explained by the fact that the functional criticality of a PE depends more on the mapping of the systolic array and less on the word size. As the mapping is the same across both the scaled versions, the fault-binning results are the same, with the slight difference due to the conversion error inherent when we downsize the PE from 32 bits to 16 bits.

Fig. 5 illustrates the classification accuracy for all the injected faults (s-a-0, s-a-1) and for all the PE buses (A, B, C, and D) of a PE in row 75, column 0 for both 16-bit and 32-bit architectures. Each bus includes bits for sign, exponent, and mantissa, according to the IEEE Std. 754. From the results, we make the following key observations:

1) Most faults in the mantissa bits for all four buses are benign, with no impact on classification accuracy.
2) The faults in the sign bits for buses A, B, and D are benign and have no impact on the classification accuracy.
3) The faults in the exponent bits for buses A and B are also benign and have no impact on classification accuracy.
4) The majority of faults in buses A and B are benign.
5) Faults in the exponent bits for buses C and D, as well as in the sign bits for bus C, are critical.

We also evaluated the functional criticality of faults in specific pins of the PE based on the fault-free range of data values passing through the pins across all iterations of the DNN. For example, for the partial-sum bus of a 32-bit PE (row 24, column 0), the maximum magnitude of the partial-sum value is less than 80 for the inferencing dataset under consideration–100 MNIST images. Thus, the maximum value of the 8-bit exponent is $127 + \lfloor \log_2 80 \rfloor$, indicating that the bits 2-5 of the exponent from the MSB-side is always 0. Hence, s-a-0 faults for these bits are always benign.

B. Analysis of Functional Criticality

Errors in buses A and B act as local perturbations to the input data stream and weights. The use of $L1$ and $L2$ regularization during training introduces sparsity in weights, making their absolute values close to 0. As the weights of the DNN are so small, error due to perturbations in input data or weights is also very small, making buses A and B benign.

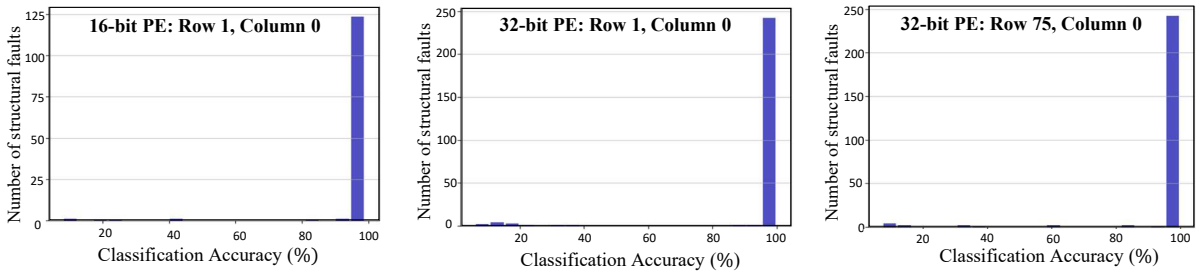

Fig. 4: Structural fault histograms generated using pin-level simulation results for different 32-bit and 16-bit PEs.

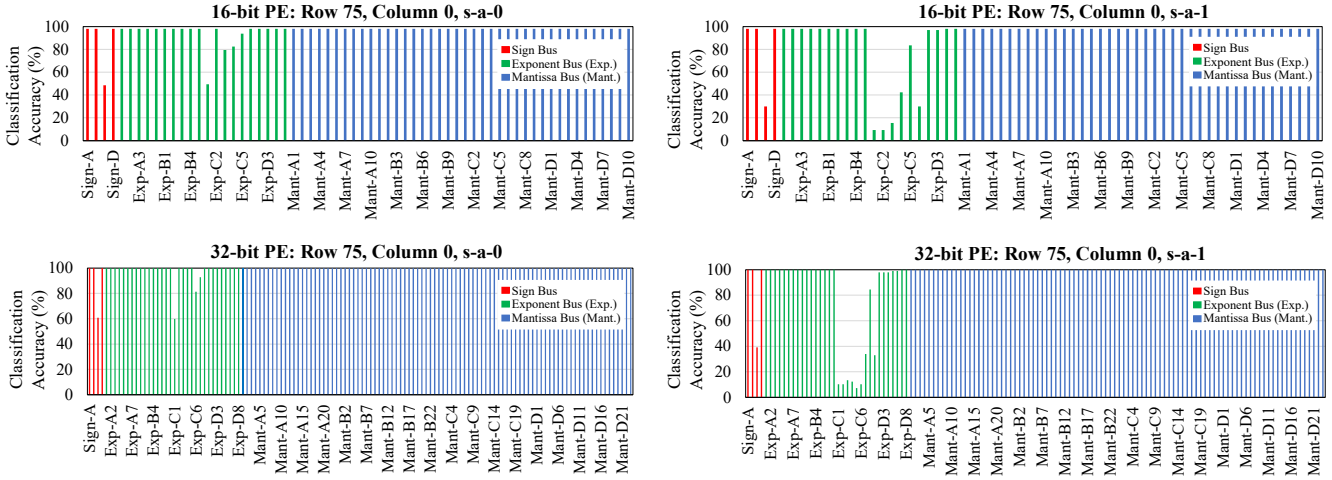

Fig. 5: Bus-level stuck-at fault classification for a PE in row 75 and column 0. Buses A and B are multiplier inputs, Bus C is the accumulation input, and Bus D is the multiplier output.

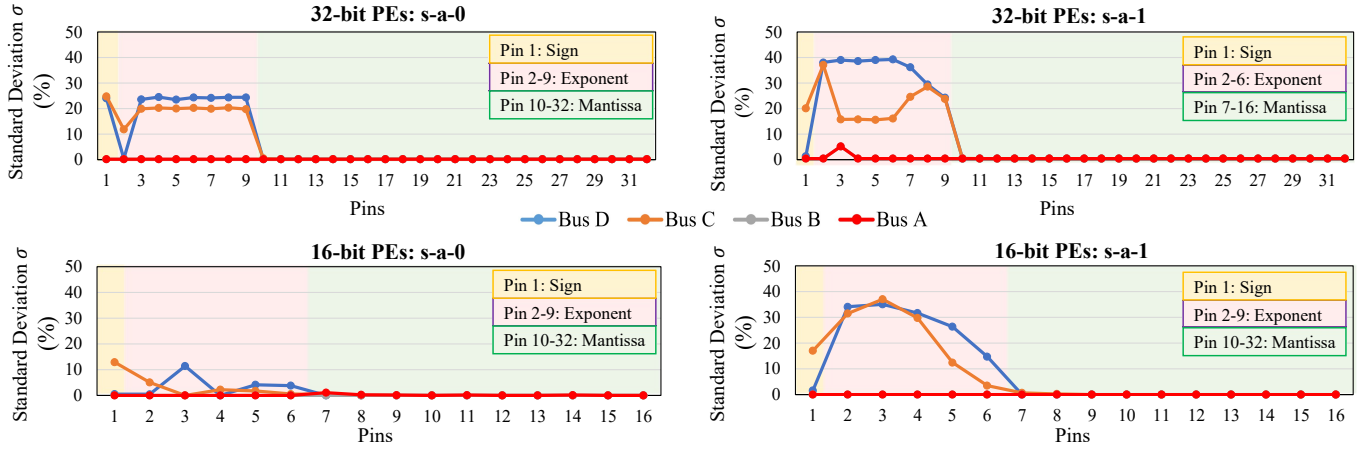

Fig. 6: Pin-level standard deviation in classification accuracy—obtained for all buses of the 85 32-bit PEs and 100 16-bit PEs spread across the systolic array, and simulated for s-a-1 faults.

Moreover, errors in these buses do not significantly affect the accumulated data across the column of PEs.

Assume that the error in weight W is ΔW and error in input data B is ΔB. As the magnitude of the weight $|W| << 1$, we conclude that ΔW is small for a single stuck-at fault. Hence, $B\Delta W$ is small in value. Similarly, as W is small, the product $W\Delta B$ is small in magnitude.

The partial sum bus C affects functional criticality, with the sign and exponent bits of bus C being the most critical. This is because C propagates the outputs accumulated by all PEs preceding it. Hence, a fault in C directly translates to a fault in the entire column of PEs; errors due to these faults are no longer local but accumulate over the entire column.

A majority of faults in the mantissa bits of a bus are benign because the magnitude of the fault-free exponent of the same bus is not high enough to amplify the error at faulty mantissa pins. Faults in the sign bits of buses A, B, and D are generally benign because their fault-free magnitudes are small.

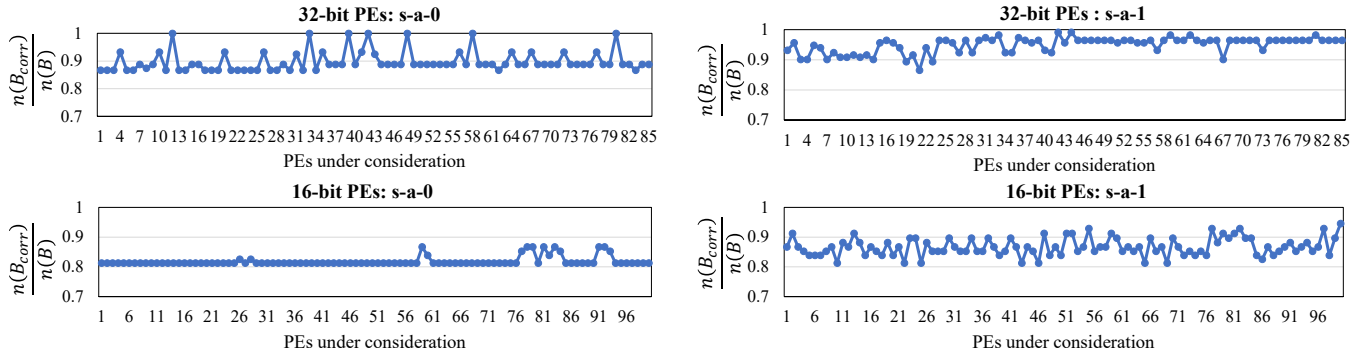

Fig. 7: Probability of benign pin-level faults (s-a-1) remaining benign in other PEs, for 16- and 32-bit architectures.

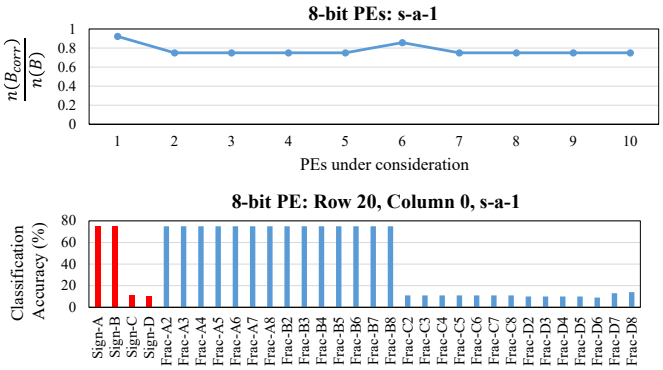

Fig. 8: (Top) Probability of a benign pin-level s-a-1 fault remaining benign across other PEs for 8-bit architecture. (Bottom) Bus-level s-a-1 fault classification for an 8-bit PE in row 20 and column 0. Similar results are obtained for s-a-0 faults.

C. Statistical Correlation among Processing Elements

We evaluated the standard deviation σ between 85 32-bit PEs and 100 16-bit PEs[1] in terms of the classification accuracy for pin-level s-a-0 and s-a-1 faults. The goal of this evaluation is to assess whether the functional criticality results for one PE can be generalized for other PEs. We used a bias-free selection strategy, choosing well-spaced rows of PEs from different columns. For column spacing, we considered the kernel size and stride of the convolutional layers to account for the overlap of data during convolution. If σ is low, we can avoid explicit fault simulation for all the PEs. In particular, since our goal is to identify and drop benign faults, low σ for benign faults is especially desirable. A low value of σ among faults located in the corresponding pins across the PEs indicates a high correlation among those faults in terms of their functional criticality.

Fig. 6 shows that the standard deviation in classification accuracy for faults in the mantissa pins is negligible for all four buses. Moreover, the σ in classification accuracy for faults on buses A and B is also low. Therefore, faults in these pins are highly correlated across PEs in the array. However, the σ for faults in the exponent and sign bits of buses C and D is significantly high. Recall that they are functionally critical, so they must be considered for structural testing.

[1]This is the number of PEs simulated at the time of paper submission.

TABLE I: Average percentage of pin-level critical faults for all four buses for 16- and 32-bit PEs.

PE Bus	32-bit PE		16-bit PE	
	% of critical faults		% of critical faults	
	s-a-0	s-a-1	s-a-0	s-a-1
A	0	0.07	0	2.18
B	0	0	0	0
C	11.06	24.37	2	15.375
D	2.05	15.10	1.93	9.375

We also evaluated the average percentage distribution of pin-level critical faults for all four buses across the 16-bit and 32-bit PEs. These results are shown in Table I. We note that bus C has the largest percentage of critical faults across all PEs under consideration, followed by bus D. Furthermore, the distribution of critical faults in these buses is localized among the sign and exponent bits.

We next analyze the likelihood of a benign pin-level fault in a given PE remaining benign in other PEs. Let B denote the set of all benign pin-level faults in a given PE. Let B_{corr} denote the set of all faults in B with low σ ($\sigma < 0.7\%$) in classification accuracy across other PEs, i.e., benign pin-level faults for the PE that are also benign in other PEs. For a pin-level fault f in the PE, let E_1 be the event $f \in B$. Let E_2 be the event $f \in B_{corr}$. Therefore, the probability that a benign pin-level fault in a PE is also benign in another PE is $P[E_2|E_1] = P[E_2 \cap E_1]/P[E_1]$. As $B_{corr} \in B$, $P[E_2|E_1] = P[E_2]/P[E_1] \approx n(B_{corr})/n(B)$, where $n(X)$ is the cardinality of set X. Fig. 7 shows the ratio $n(B_{corr})/n(B)$ for pin-level faults in both 16- and 32-bit PEs. This ratio exceeds 0.9 in most cases, indicating a high likelihood that a benign pin-level fault in a PE is also benign in other PEs. Fig. 8 shows similar results for both the criticality of pin-level faults and the distribution of benign faults for the 8-bit PE architecture.

D. Criticality of Multiple Pin-Level Faults

Fault simulation was also carried out for multiple pin-level faults, i.e., multiple stuck-at faults in the buses in the PE. We made the following key observations: (1) Multiple stuck-at faults in buses A and B are all benign. Irrespective of how many bits are stuck at 0 or 1, we always get 100% accuracy; (2) Multiple stuck-at faults in the exponent bits of buses C and D are critical; (3) Multiple stuck-at faults in the mantissa

bits of buses C and D are benign.

We note that large groups of multiple stuck-at faults, especially in the mantissa pins, in the output buses of adder and multiplier are functionally benign. The pin-level criticality information based on multiple pin-level fault simulations can be potentially exploited to derive knowledge of the criticality of single stuck-at faults internal to the adder and multiplier. We present two theorems, which state how the criticality of internal stuck-at faults is implied by pin-level fault criticality. Proofs of the theorems are available in [**?**] and these theorems are utilized in Section V.

Theorem 1. *Let A represent the scenario of concurrent (multiple) s-a-0 faults on all the output stems of a set of nodes S in the PE. Let B represent the scenario of concurrent s-a-1 faults on all the output stems of the nodes in S. If both A and B are functionally benign under a criticality threshold ϵ, any combination of concurrent s-a-0 faults, s-a-1 faults, and erroneous bit-flips in the outputs of the nodes in S is also functionally benign under ϵ.*

Theorem 2. *Let all combinations of stuck-at faults (single and multiple) on a set of nodes S in a PE be functionally benign. Then any internal stuck-at fault whose effect propagates only to nodes in S is also functionally benign.*

V. Greedy Iterative Method for Determining Fault Criticality

A. Exclusivity-driven Greedy Iterative Framework

Let S_F indicate a set of nodes in the netlist. The exclusive fan-in cone of S_F contains circuit nodes which are in the fan-in cones of the nodes in S_F but not in the fan-in cones of nodes outside S_F. The number of nodes in the exclusive fan-in cone of a set of nodes, S_F, is referred to as the *exclusivity* of S_F: $EXCFI(S_F)$. For example, let the exclusive fan-in cones of nodes X and Y be the sets S_X and S_Y, respectively; this implies: $S_X \cap S_Y = \emptyset$. The exclusive fan-in cone of the set of nodes, $S_F = \{X, Y\}$, is a super-set of $S_X \cup S_Y = S_X + S_Y$. Hence, $EXCFI(S_F) \geq EXCFI(X) + EXCFI(Y)$.

We convert the PE netlist into a directed graph G and levelize it. In G, vertices represent logic gates and edges represent wires. During graph levelization, the vertices in a graph are distributed among *levels* which are defined as follows. Let there be n_g levels in a graph — $\{L_1, L_2, ..., L_{n_g}\}$ — where L_1 indicates the level containing the output vertices, i.e., vertices with no outgoing edges. While traversing the levelized graph starting from its input vertices, nodes in L_i are always encountered before nodes in L_j, where $i > j$. A graph can be levelized if it does not contain cycles.

In the pin-level functional criticality experiments on the adder and multiplier, the pins were essentially output stems of the output vertices or nodes of the circuit constituting level L_1 of G. The principle of the greedy iterative criticality classification framework is to traverse up the levels of G, starting from L_1, and while doing so, perform fault simulation of nodes in each level of the graph. This level-wise fault simulation effectively treats the output stems of a certain level's nodes as the new "pins".

```
GREEDY(G, L_sim, S_benign) :
Input: G, L_sim, S_benign
Output: L_m ← level with highest potential exclusivity
//Array EXC contains node-exclusivity sums for different
  levels in G;
Initialize EXC to 0 for levels ∉ L_sim;
for L_i ∉ L_sim do
    for v ∈ L_i do
        if v ∉ S_benign then
            EXC[L_i] ← EXC[L_i] + EXCFI(v);
        end
    end
end
L_m ← L_i(∉ L_sim) with maximum EXC[L_i];
```

Fig. 9: Procedure for greedy level selection.

Similar to the criticality implication of internal faults from pin-fault criticality, Theorem 2 is then used to identify functionally benign nodes in the exclusive fan-in cone of a set of nodes that are found to be functionally benign when both concurrently s-a-0 and s-a-1. As the levels are traversed, more benign nodes are identified iteratively; this leads to the simulation of fewer nodes with unknown criticality in a higher level in a later iteration. In one iteration, all the nodes in one graph level are simulated: either one-by-one (single fault simulations) or group-wise (multiple fault simulations), based on user discretion. If a node or group of nodes is found to be benign, all nodes in its exclusive fan-in cone are marked as benign. Group-wise fault simulations are more likely to identify higher number of benign nodes as compared to single-fault simulations. This is because the exclusivity of a benign group of nodes is greater than or equal to that of a single benign node. According to Theorem 1, if a group of nodes is benign when concurrently s-a-0 and s-a-1, individual nodes in that group are also benign. However, if a group of nodes is functionally critical, we cannot determine the criticality of individual nodes without simulating them one by one.

The greedy aspect of the methodology consists in selecting the level of the graph, which has not been simulated so far, with the highest potential exclusivity after each iteration. It means that we simulate the nodes of a level that shows the promise of identifying the maximum number of benign nodes in that iteration, thereby reducing the total number of simulation runs possibly required to determine the criticality of all the nodes in the netlist. Therefore, during each iteration, we select the level (from the remaining levels not simulated so far) with the highest value of $\sum_{j=1}^{N} EXCFI(v_j)$, where v_j is a node in the level containing N nodes.

The procedure, $GREEDY()$, for greedy level selection in the PE netlist is described in Fig. 9. Here, S_{benign} denotes the set of benign nodes already identified in the adder and L_{sim} denotes the set of levels already simulated. The level, L_m, with the highest exclusivity is returned as the output of $GREEDY(G, L_{sim}, S_{benign})$. The flowchart for the greedy iterative criticality classification framework with group-wise

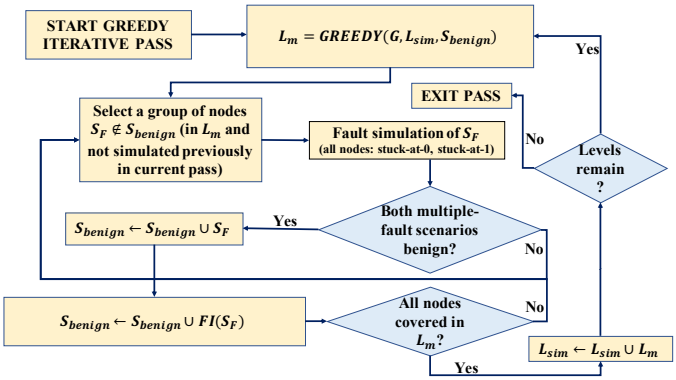

Fig. 10: Greedy framework for identifying benign nodes.

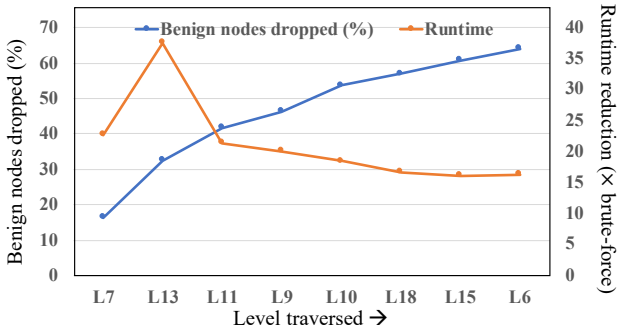

Fig. 11: Cumulative determination of benign nodes in the 32-bit adder netlist of PE(20,0) by the greedy iterative framework.

node simulations is illustrated in Fig. 10. The framework uses $GREEDY()$ and $FI()$ procedures to identify benign nodes in the adder. The procedure $FI(S_F)$ returns the set of nodes in the exclusive fan-in cone of the set of nodes S_F. In the first pass, we traverse through all the levels of G greedily; for each level's nodes, we perform group-wise (multiple) fault simulations. The functionally benign groups, along with nodes in their exclusive fan-in cones, are added to the set of known benign nodes, S_{benign}. The criticality of individual nodes in the functionally critical groups remain unknown. After all the levels have been traversed in the first pass, the second pass begins with the updated S_{benign} and L_{sim} reset to $\emptyset$. The decision to proceed with multiple passes depends on whether a sufficient number of nodes have been classified as benign.

B. Evaluation

The number of nodes in the group S_F varies based on the number of nodes in a given level. We fix the number of groups as 10. Nodes in a level are equally distributed among the 10 groups; nodes with higher exclusivity are grouped together. We execute the first pass of the greedy framework to identify benign nodes in the netlist of 32-bit PE(20,0).

Fig. 11 shows the cumulative number of benign nodes in the 32-bit adder of PE(20,0) (expressed as a percentage of the total node count) identified during a single pass of the greedy iterative framework; the ratio of cumulative runtime of brute-force fault simulation to that of the greedy method is also shown. The x-axis indicates the order in which the levels in the adder are traversed by the framework. For example, $L13$ implies $L_m = 13$ in that iteration of the framework. There are a total of 21 levels in the adder. Benign nodes identified by the first eight iterations of the first greedy pass are reported; more than 60% of nodes in 32-bit adder are marked as benign. For the 32-bit multiplier, nearly 64% of the nodes are marked as benign after 11 iterations (levels) of the first pass of the greedy framework. There are 15 levels in the multiplier netlist.

The number of benign fault sites exceeds the number of critical fault sites in the PE netlist due to the inherent fault tolerance associated with the end-application (image classification). Consequently, it is computationally expensive to uncover all critical fault sites in the PE through brute-force fault simulations. The greedy iterative framework leverages Theorems 1–2, and enables an accelerated search for the critical fault locations in the PE by identifying more benign locations using fewer fault simulations. However, the scalability of this method is restricted by the following fundamental limitations that extend to any criticality evaluation approach based on explicit fault simulations.

- The framework cannot determine functionally critical nodes unless single stuck-at faults are separately simulated for the nodes under consideration. Multiple (concurrent) faults in a group of nodes can be functionally critical even if single faults on individual nodes are benign. Therefore, inferring the criticality of single stuck-at faults from the criticality of the corresponding multiple-fault scenario is possible if and only if the latter is benign.
- If faults in a node X are benign, only the faults in the exclusive fan-in cone of X are going to be benign; see Theorem 2. As the sizes of exclusive fan-in cones depend on the netlist topology, there may exist a PE netlist for which the greedy framework determines only a small number of benign nodes during its initial passes. As a result, more passes will be required to identify sufficient benign nodes and the runtime will approach that of exhaustive fault simulations.
- There is no optimal way to decide the size of the group of nodes S_F for multiple fault simulations. If the size is kept large, the group is very likely to be critical and has to be decomposed into smaller groups for criticality evaluation. On the other hand, if the group size is kept small, the number of fault simulations (and runtime) increases.
- The distribution of nodes in groups is a factor in determining the number of benign nodes identified in a given number of fault simulations. The total number of ways in which n nodes in a level can be distributed into groups of size m are $\prod_{k=0}^{\lceil n/m \rceil} \binom{n-km}{m}$. Therefore, design-space exploration for selecting the optimal node distribution is computationally intractable.

C. Functional Criticality Analysis using Data-Driven Approaches

We have shown that pin-level fault-criticality is highly correlated across PEs in the array, indicating a possibility of the criticality being correlated to the physical location of

the fault. It is also evident that functional criticality strongly depends on the application workload (i.e., inferencing image-set), parameters of the mapped DNN model, and DNN-to-array mapping. Powerful statistical and data-driven methods can then be unleashed to learn the underlying correlation between functional criticality and the various attributes of a structural fault. For example, a machine-learning classifier can potentially extract the complex non-linear relationships between functional fault criticality and the various application-related, mapped model-related, and netlist-related factors that affect criticality. As a result, the criticality of a large number of faults in the array can be potentially inferred without explicit fault simulations, by utilizing the criticality knowledge of fewer other faults, thereby reducing the required number of fault simulations.

We next present a scalable ML-based framework for criticality evaluation. The greedy iterative framework is effective in systematically uncovering a large number of functionally benign faults with a small number of gate-level fault simulations. The ML-based framework integrates the greedy framework to collect criticality information (ground-truth) for a limited number of nodes in a PE for training the ML models.

VI. Structural Fault Criticality Assessment Using Machine Learning

If both s-a-0 and s-a-1 faults on the output stem (fault site) of a logic gate (node) X are functionally benign, X is also classified as functionally benign. Otherwise, X is classified as functionally critical. Internal fault sites or nodes that lie in the exclusive fan-in cone of benign pins are provably benign (Theorems 1-2). However, the number of such nodes internal to the PE is only 6% and 9.4% of the total number of nodes in 32-bit and 16-bit PEs, respectively. The remaining faults are only potentially benign because the errors due to faults in those nodes may simultaneously propagate to both benign (mantissa) and critical (sign and exponent) pins. As described in Section II, the use of brute-force simulations to evaluate the criticality of these PE-internal faults is computationally impractical. Section V describes the limitations of all methods that depend solely on explicit gate-level fault simulations for estimating the functional criticality of structural faults.

In this section, we explore a computationally efficient ML-based method to determine functional criticality. This method can scale to large PE arrays, a large number of faults, and a high volume of input data. The proposed approach is based on the use of deep learning to predict fault criticality. ML-based models are known to be adept at learning hidden correlations when they are trained with relevant features and data [26]. We propose effective deep-learning strategies that take into consideration the fault-free data stream, the (global) circuit topology, and (local) information about the fault sites to enable the accurate classification of fault criticality. They utilize structural and dataflow features from the design for model training. Furthermore, a generative adversarial network (GAN) is trained to prevent the deep-learning models from misclassifying critical faults as benign ("test escape").

A. Feature Selection for Training ML Model

The training data consists of several sets of features obtained from strategically sampled fault sites within the gate-level netlist. As the functional criticality of structural faults depends on both the circuit topology and the fault-free dataflow during functional simulation, we ensured that the selected features reflected both topology and dataflow.

1) Topological Feature Selection

For a given gate-level netlist, the topology-based features include the depth of the node X (i.e., the fault site), $depth_{PO}$, from the nearest primary output (PO); depth of X from the nearest primary input (PI), $depth_{PI}$; number of PIs in the fan-in cone of X, $count_{PI}$; number of gates in the fan-in cone of X, $count_{fi}$; number of gates in the fan-out cone of X, $count_{fo}$; number of sign and exponent pins (combined) in the fan-out cone of X, $f_{critical}$; number of mantissa pins in the fan-out cone of X, f_{benign}. Finally, a set of features f_{gates} is added to consider the frequency of occurrence of various standard cells in the fan-in and fan-out cones of X.

2) Dataflow-based Feature Selection

Dataflow-based features are computed based on weighted compression of the fault-free bit streams appearing at each node across all computation cycles and across all image classes in the inferencing dataset. The total number of bit streams equals the number of test images, T_{im}. The length of each bit stream is equal to the total cycle count for inferencing, N_{cyc}. As the number of bit streams scales with the number of images in the inferencing dataset, it is possible to first compress those bit streams across the T_{im} images in the dataset through majority counting per class, i.e., choosing the bit-value that occurs the maximum number of times across all images belonging to a particular class in the dataset, followed by weighted compression along the N_{cyc} cycles. Compression across the T_{im} images reduces the number of bit streams from T_{im} to 10 for MNIST dataset, ensuring one representative bit stream per class; T_{im}=100 in this work. However, majority-based compression leads to loss of information and does not account for misclassifications which result from the minority bit streams, leading to ineffective representation of the critical information in the dataflow feature vector. In order to avoid this situation, it is essential to take bit streams corresponding to all input images into account during dataflow-based feature extraction.

The bit streams associated with all images in the test dataset are then compressed along the N_{cyc} cycles through weighted compression. Suppose that $b_{i,j}$ is the i^{th} bit of the j^{th} bit stream, then the weighted score for that bit stream is $S_j = \sum_{i=1}^{N_{cyc}} b_{i,j} \cdot i$. This cycle-based weighting is done to ensure that each cycle is uniquely represented with higher weightage given to cycles closer to the end of the computation, i.e., the output prediction stage. Following this step, we have 100 dataflow-based features per node—corresponding to each input image.

The training data for the ML model is obtained by sampling several fault sites, performing fault simulation to obtain

their functional criticality, and extracting their topology- and dataflow-based features. We partition the collected data points into training, testing, and validation sets, and ensuring that each level of the circuit is uniformly represented. To select data points for training, we first levelize the gate-level netlist. Then we simulate groups of nodes in different topological levels of the netlist with all nodes in a group having concurrent s-a-0 or s-a-1 faults. For the benign groups obtained through this process, nodes lying in their exclusive fan-in cones are also provably benign according to Theorems 1-2. Hence, more benign nodes are identified with fewer fault simulations using group-wise multiple-fault simulations. The critical groups are further broken down and simulated one node at a time to uncover critical nodes. The number of benign nodes generally exceeds the number of critical nodes owing to the intrinsic fault tolerance of the accelerator array. Once the ratio of the number of benign nodes to that of the critical nodes falls below 6, we add the collected nodes to the training dataset in such a way that all topological levels in the netlist are adequately represented in the training data.

B. Fault-Criticality Assessment

We utilize DNN-based classifiers and the features described in Section V-A for assessing the functional criticality of stuck-at faults. We use a combination of convolutional layers followed by fully connected layers as the network architecture. We use this architecture because convolutional layers allow us to perform additional feature-space exploration on top of the features already supplied during training. The fully connected layers are used for classification as the final step.

Motivated by the regular structure of the systolic array, we select a single PE for model training. Two separate models with identical architectures; are trained for sub-blocks within the 32-bit PE, one for the 32-bit floating-point adder and the other for the 32-bit floating-point multiplier unit. Table II summarizes the fault-criticality classification results based on different ratios of dataset partitioning for training and evaluating the DNN model. A DNN configuration with n convolutional layers and m fully connected layers is denoted as [nC,mF]. The first fully connected layer, immediately following the convolutional layers, contains 271 neurons while the final fully connected layer contains two neurons as we are performing binary classification of node criticality—critical or benign. Faults in 786 (661) nodes of the 32-bit adder (32-bit multiplier) in PE(20,0) were simulated to obtain the corresponding node criticality information; r_{tr} denotes the fraction of the simulated nodes used for training the DNN model whereas the remaining are used for evaluating the trained model. For both the 32-bit adder and the 32-bit multiplier, the [0C,7F] configuration provided the highest mean test accuracy and the lowest mean test-escape compared to other alternatives that we considered. Therefore, we select [0C,7F] as the DNN architecture for fault-criticality assessment.

TABLE II: Results obtained using DNN for fault-criticality assessment in 32-bit adder and multiplier.

Netlist	r_{tr}	A_{ML} (%)	Mean A_{ML} (%)	TE (F_{cr})	Mean TE/F_{cr} (%)
Adder	0.6	89.8	90.8	22 (79)	24.1
	0.7	92.3		13 (57)	
	0.8	90.4		10 (35)	
Multiplier	0.6	94.6	95.2	6 (27)	25.0
	0.7	93.9		9 (25)	
	0.8	97.1		2 (12)	

r_{tr}: fraction of the collected data used in training; A_{ML}: prediction accuracy; TE: number of critical faults misclassified as benign (test escape); F_{cr}: number of critical faults in evaluation set.

C. Reducing Misclassifications with Adversarial Learning

A challenge with fault-criticality assessment using DNN is the inevitability of misclassification; even a highly effective model can sometimes lead to misclassification. Not surprisingly, this is also the case with our classification method, as shown in Table II. While the occasional misclassification of benign faults as critical is acceptable, the classification of critical faults as benign ("test escape") is of greater concern. Therefore, we have developed a second ML model (ML2) that can be trained to identify misclassifications from the nodes predicted as benign by the first DNN model (ML1).

The low misclassification rate of ML1 is a challenge because it provides ML2 with only a limited amount of training data. We utilize a generative adversarial network (GAN) [27] to overcome this performance bottleneck. GANs are powerful tools that can be utilized to learn the probability distribution of a specified dataset [28]. We exploit this property of adversarial training to generate test-escape data for training ML2. The overall methodology for training the aggregated GAN based criticality assessment flow is illustrated in Fig. 12.

D. Evaluation

Training and Validation: Ground-truth data comprising the criticality information of nodes in a gate-level PE is collected for training and validation of ML1. Single fault simulation is carried out to determine the functional criticality of a node based on a pre-determined threshold. We use the greedy iterative framework to collect benign nodes and to mark the critical node-groups. Faults in critical node-groups are individually simulated to obtain critical nodes.

We set a conservative threshold of 95% classification accuracy for collecting more critical-labeled nodes; training the ML models on a large number of critical nodes will lead to low test escape during criticality evaluation. While we demonstrate the ML1-ML2 flow for binary classification of fault criticality, i.e., benign and critical, the proposed method can be extended to more classes (levels) of fault criticality.

The above set of nodes is randomly partitioned into training and validation datasets in the ratio 4:1. For each such partitioning scenario, ML1 is first trained to classify nodes as critical or benign. The trained model is then evaluated on the validation set. The critical nodes in the validation set that are misclassified as benign by ML1 are added to the set of misclassified nodes S_{TE}. During each iteration of training and validation, S_{TE} is augmented with misclassified nodes from the predictions of ML1 for the critical nodes in the

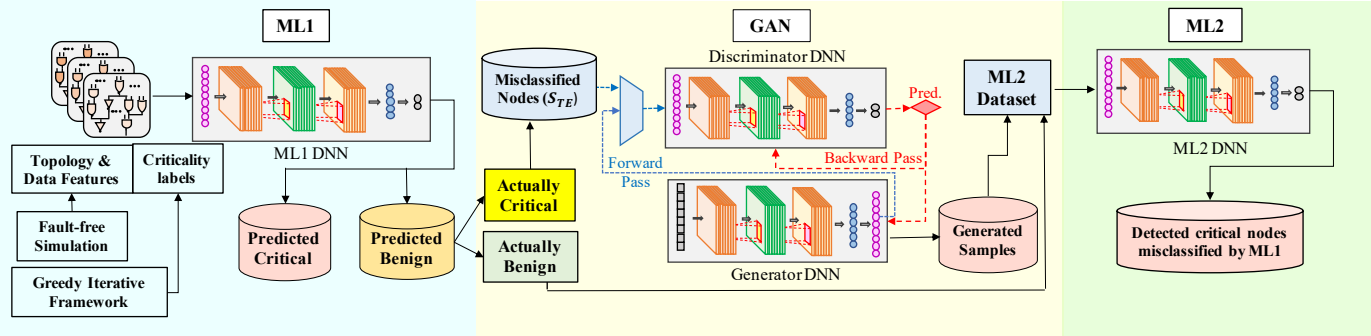

Fig. 12: Methodology used for training aggregated GAN-based criticality assessment flow.

validation set. Each such iteration also produces a separately trained version of ML1. After a total of $N_{tr,val}$ training-and-validation iterations ($N_{tr,val}$ is pre-defined), the version giving the highest classification accuracy on the corresponding validation set V_{best} is selected for evaluation. The set S_{TE}, accumulated across $N_{tr,val}$ iterations, is used for training the GAN to generate samples with features matching the feature distribution of the actual misclassified nodes. This ML scheme is referred to as aggregated GAN-based criticality assessment. To assess the convergence of GAN after 1000 epochs, we take GAN-generated samples and feed them to ML1 for prediction. If ML1 predicts all the samples to be benign, we conclude that the GAN has converged and is able to generate samples with features imitating those of real critical nodes that were misclassified as benign by ML1. The convergence of GAN can be achieved by: (i) increasing the number of training epochs, (ii) tuning the architecture and hyperparameters of GAN, and (iii) increasing $N_{tr,val}$. After the GAN converges, the trained generator model in the GAN is ready for generating test escape-like samples to further augment S_{TE}. The augmented S_{TE}, along with nodes in V_{best} correctly classified as benign by ML1, is used to train ML2, to distinguish between real and misclassified benign nodes.

Evaluation: During the evaluation phase of aggregated GAN-based criticality assessment flow, fault simulation of nodes, not present in the training and validation sets, is carried out to form the evaluation dataset T. The pre-trained ML1 model is used to predict the criticality of nodes in T. The nodes predicted as benign by ML1 are fed to the pre-trained ML2 model for post-processing—ML2 evaluates if any of those nodes is misclassified. Finally, the nodes classified as real benign by ML2 are categorized as the truly-benign nodes.

E. Benefits of 2-Tier Framework driven by Data Augmentation

We train the first tier of our 2-tier criticality analysis framework, i.e., ML1, on a larger data pool that contains all benign and critical faults obtained during ground-truth collection. The second tier, i.e., ML2, is trained on a smaller and targeted data pool containing benign faults and critical faults that were mispredicted as benign by ML1, i.e., test escapes. These test escapes resemble benign characteristics that led to their misclassification by ML1. No additional features are extracted for training ML2 and the model architectures for ML1 and ML2 are identical. The objective of ML2 is to identify the properties of the test-escape faults that led to their misclassification by ML1; in other words, ML2 learns from the mistakes made by ML1.

ML1 learns on a more diverse data pool with higher intra-dataset variance compared to ML2 which learns on a more targeted dataset. As a result, the non-linear class boundary learnt by ML1 separates a majority of benign faults from critical faults. Unfortunately, the same class boundary misclassifies some critical faults as benign because the critical faults' feature distribution closely matches the benign faults' feature distribution. Now, when we include only the misclassified critical faults from ML1 along with benign faults in the training dataset of ML2, ML2 can learn a new class boundary that is able to separate benign faults from the relatively few test escapes.

The motivation behind our 2-tier framework is similar to that of boosting-based ensemble methods in machine learning. These methods deploy cascaded models where a successor model learns on a modified dataset and corrects errors in the predictions of its predecessor [29]. Fig. 13 shows the feature distributions of benign and critical faults in the training datasets for ML1 and ML2 to illustrate the contribution of dataset modification to the superior performance of the 2-tier framework with no additional feature extraction for ML2. In Table III, we evaluate the performance benefits of the 2-tier framework. Prior work on using specialized DfT architecture for efficient structural testing of systolic array based accelerators can leverage the proposed criticality analyzer for targeted testing of critical faults [30].

The objective of GAN is to augment the test-escape data for ML2 because ML1's test-escape is typically very low and not sufficient to train ML2. Finding such test-escape-like faults in the PE netlist using gate-level fault simulations is time-consuming. We may potentially end up simulating a large majority of faults in the PE before encountering such an example. This situation will defeat the purpose of having an efficient framework that can predict fault criticality for the entire PE by using a small amount of labeled criticality data. A dataset of any size, as desired by the user, can be quickly generated by GAN. The usage of GAN also makes the 2-tier framework robust by increasing the amount of diversity in its training samples.

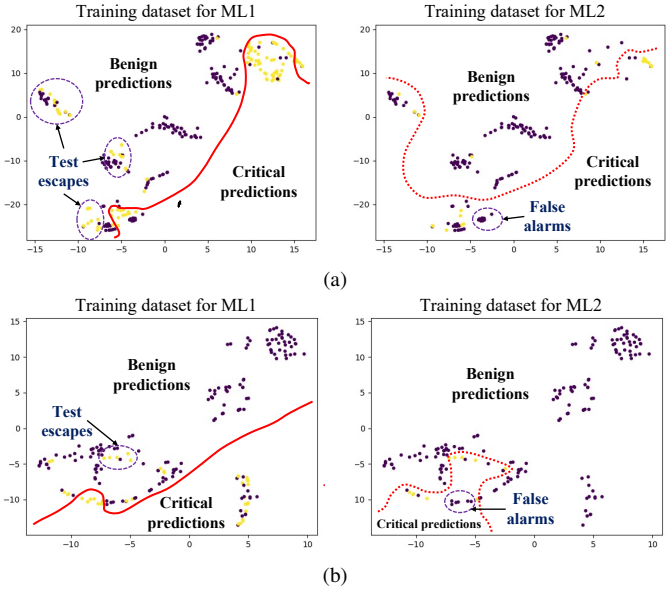

Fig. 13: Class boundaries learnt by ML1 (left) and ML2 (right) for: (a) 32-bit adder in PE(20,0) and (b) 32-bit multiplier in PE(20,0). Benign and critical faults are indicated by blue and yellow dots, respectively. Class boundary learnt by ML1 (ML2) is shown by the solid (dotted) red line.

VII. Experimental Results

A. Experimental Setup

The aggregated GAN-based flow is used to evaluate the functional criticality of nodes in the 32-bit adder and multiplier netlists of PE(20,0), and the entire netlist of 16-bit PE(20,0). The ML models are implemented using CUDA-enabled PyTorch. Functional fault simulation for ground-truth collection takes 58 hours per 100 nodes on an Intel Xeon E5 2.4 GHz CPU (45 parallel processes). Feature extraction for 100 nodes takes ∼2.5 hours each for the 32-bit adder, 32-bit multiplier, and 16-bit PE. While training data collection is time consuming, it is a one time effort. The number of benign (critical) nodes for training and validation of the 32-bit adder, 32-bit multiplier, and 16-bit PE are 713 (207), 477 (77), and 224 (116), respectively. The number of benign (critical) nodes in the evaluation sets for the 32-bit adder, 32-bit multiplier, and 16-bit PE are 251 (125), 228 (42), and 331 (182), respectively. The feature dimensionalities for both 16-bit and 32-bit PEs are 277. Seven iterations of training and validation ($N_{tr,val} = 7$) are performed to form S_{TE} for training the GAN and for selecting the best-trained ML1 model for the 32-bit adder, 32-bit multiplier, and 16-bit PE. It takes ∼3 min for training and validation of the aggregated GAN-based flow.

The ML1-ML2 evaluation flow is very quick, taking ∼0.1 s for 100 nodes. For a 8-bit PE with 240 nodes, brute-force fault simulation takes 139.2 hours. Our pre-trained 2-tier framework completes the criticality evaluation of the entire 8-bit PE in only 0.5 s, resulting in a $10^6\times$ speedup in PE-level criticality analysis. The runtime improvement scales significantly during functional criticality analysis of the entire systolic array.

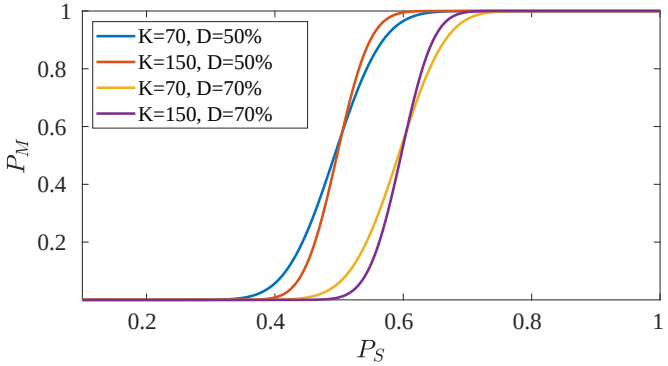

Fig. 14: Probability of correct decision during multi-frame classification.

B. Threshold Analysis for Catastrophic Test Escape

Let P_S be the inferencing accuracy of a DNN accelerator. Let K be the number of frames used by an autonomous system, such as a self-driving car, to take a decision on the object classification [31]. Let $D_{S,T}$ be the decision threshold, i.e., the decision taken by the autonomous system is determined by the class that is predicted by DNN for at least $D_{S,T}\%$ of the K frames captured per second by the camera.

The probability that a single frame is correctly classified by the DNN is equal to P_S. The probability P_M that a correct decision is taken by the system based on the classifications of the K frames is equal to the probability that at least $D_{S,T}\%$ of the K frames are correctly classified. In other words, $P_M = \sum_{i=\lceil DK/100 \rceil}^{K} \binom{K}{i} P_S^i \cdot (1-P_S)^{K-i}$. The value of K typically exceeds 150 [31]. Fig. 14 shows the variation of P_M as P_S is varied from 0.1 to 1 for $D \in \{50, 70\}$ and for $K \in \{70, 150\}$. The value of P_S corresponding to the saturation points of all four curves is ∼0.8, i.e., the minimum acceptable single-frame accuracy of the accelerator can be set to 80%. Faults that cause less than 80% inferencing accuracy are considered as *catastrophically critical* faults. The catastrophically critical faults misclassified as benign are considered as *catastrophic test escapes*. The percentage of such faults (out of the total number of critical faults in the evaluation set) after applying ML1 (ML2) is quantified by TE_1^{cat} (TE_2^{cat}).

C. Evaluation of Fault Criticality

Table III shows the evaluation results for the 32-bit adder and multiplier in PE(20,0), 16-bit PE(20,0), and 8-bit PE(20,0). In each experiment, the 2-tier ML1-GAN-ML2 framework is trained and validated from scratch using a different random seed for weight initialization, followed by evaluation on the dataset T. Both s-a-0 and s-a-1 faults in a benign node are benign. On the contrary, faults in a critical node are not necessarily catastrophically critical; we individually analyze the faults in a critical node misclassified as benign by ML1 (ML2) to obtain TE_1^{cat} (TE_2^{cat}). Thus, the s-a-0 and s-a-1 faults in each misclassified node, after evaluation by ML1 and ML2, are individually analyzed and

TABLE III: Evaluation of 2-tier framework trained and validated on the netlists of 32-bit PE(20,0), 16-bit PE(20,0), and 8-bit PE(20,0).

Experiment	Adder in 32-bit PE				Multiplier in 32-bit PE				16-bit PE				8-bit PE			
	A_{ML1} (%)	A_{ML2} (%)	TE_1^{cat} (%)	TE_2^{cat} (%)	A_{ML1} (%)	A_{ML2} (%)	TE_1^{cat} (%)	TE_2^{cat} (%)	A_{ML1} (%)	A_{ML2} (%)	TE_1^{cat} (%)	TE_2^{cat} (%)	A_{ML1} (%)	A_{ML2} (%)	TE_1^{cat} (%)	TE_2^{cat} (%)
I	92.4	92.6	6.7	3	85.6	85.6	2.3	2.3	73.1	73.1	0.7	0.6	80.6	80.6	0	0
II	85.1	85.1	3	3	89.3	90.4	7.1	2.3	73.1	73.1	0.6	0.6	80.8	80.8	0	0
III	91.2	91	3	3	85.9	85.9	2.3	2.3	79.1	80.1	1.2	0.6	93.5	90.3	0	0
IV	87.1	87.3	3.6	3	85.2	86.3	11.9	4.7	77.8	77.8	0.7	0.7	80.6	80.6	5	5
Average	89	89	4.1	3	86.5	87	5.9	2.9	75.8	76	0.8	0.4	83.9	83.1	1.2	1.2

TABLE IV: Evaluation of 2-tier framework trained and validated on the netlists of 32-bit PE(20,0), 16-bit PE(20,0), and 8-bit PE(20,0) for different criticality thresholds.

Δ_{th} (%)	Adder in 32-bit PE				Multiplier in 32-bit PE				16-bit PE				8-bit PE			
	A_{ML1} (%)	A_{ML2} (%)	TE_1 (%)	TE_2 (%)	A_{ML1} (%)	A_{ML2} (%)	TE_1 (%)	TE_2 (%)	A_{ML1} (%)	A_{ML2} (%)	TE_1 (%)	TE_2 (%)	A_{ML1} (%)	A_{ML2} (%)	TE_1 (%)	TE_2 (%)
3	91.6	91.6	6.6	6.6	77.8	77.8	4.7	2.3	52.4	52.4	0	0	84.6	87.2	3.7	0
2	94	92.8	4.1	4.1	86.7	86.7	4.5	4.5	68.6	68.6	0	0	73.1	73.1	0	0
1	92.2	92.2	7.4	6.8	89.2	64.4	13.3	8.8	80.9	80.9	0	0	91.3	91.3	0	0

deemed to be catastrophic if they cause more than 20% drop in accuracy from the fault-free inferencing accuracy. As the fault-free inferencing accuracy of the 8-bit fixed-point architecture is 75%, the threshold for catastrophic criticality is set to 68%.

The accuracy A_{ML2} of the 2-tier framework is not very high in all cases because the minimization of test escape after ML2 is deliberately given higher priority than A_{ML1}. The relatively lower A_{ML1} in different iterations of training and validation help to accumulate valuable test-escape data for training the GAN and ML2. Thus, the ML1-ML2 combination effectively reduces the test-escape count during criticality assessment. We find that TE_2^{cat} is 3% on average for the criticality assessment of the 32-bit adder. For the 16-bit PE (32-bit multiplier), TE_{cat} is 0.4% (2.9%) on average. Therefore, we conclude that an overwhelming majority of the misclassified faults do not cause significant errors in the inferencing behavior of the accelerator.

We also evaluate the effectiveness of our framework in criticality classification when the criticality threshold is set close to the fault-free inferencing accuracy by the end-user. The fault-free inferencing accuracies for 8-bit fixed-point, 16-bit floating-point, and 32-bit floating-point architectures are 75%, 98%, and 99%, respectively, on the MNIST workload comprising 100 images. The threshold drop in accelerator's inferencing accuracy (Δ_{th}) due to a critical fault is set to $\{1\%, 2\%, 3\%\}$. Table IV shows the criticality evaluation results for all three PE architectures at the higher criticality thresholds. The average test-escape is 5.5%, 0%, and 0% for 32-bit, 16-bit, and 8-bit PEs.

D. Transferability of ML Method

Results from Section IV-B indicate a high correlation among benign pin-level faults across several PEs. Fig. 7 shows that it is extremely unlikely for a benign fault in a given PE to become critical in another PE, providing preliminary insights into the transferability of ML models. If the ML models are trained on PE Y, the trained models can evaluate fault-criticality in a different PE Z with low test escape, i.e., the models are *transferable*, if: (i) the ML models are conservatively trained on a large number of critical nodes in Y, and (ii) majority of the benign nodes in Y remain benign in Z. If (i) is satisfied, the trained models will predict nodes in Z with similar features to be critical, although some of the critical nodes of Y may become benign in Z due to poor correlation. The test escape, however, is low because of the conservative training undertaken. If (ii) holds as hinted at by Fig. 7, very few benign nodes in Y will become critical in Z. Therefore, although ML models trained on the benign nodes in Y may misclassify the benign-turned-critical nodes in Z as benign, the number of such misclassifications is extremely low.

The disparity between high-dimensional feature distributions of nodes in two PEs can be measured using a statistical metric called *maximum mean discrepancy* (MMD) [32]. The MMD between two feature distributions is given by the reproducing kernel Hilbert space (RKHS) based distance between the mean features of the two distributions. Higher value of MMD indicates larger separation between the two feature distributions which, in turn, implies lower likelihood of model transferability across PEs. We use the radial-basis function kernel to define the RKHS.

Table V highlights the transferability of ML models trained on the adder and multiplier of 32-bit PE(20,0), and 16-bit PE(20,0), i.e., *source* PEs. These models are used to assess the functional criticality of faults in five other PEs, i.e., *target* PEs. We further transfer the trained framework to evaluate fault criticality in PE(20,0) and PE(45,8) for a different inferencing workload of 100 MNIST images. The MMD between the source S and target T feature distributions is denoted by $D_{S,T}$. We also compute the mean $\mu(D_{S,S})$ and standard deviation $\sigma(D_{S,S})$ of MMD between two distributions randomly sampled from the same source feature distribution S. While the values of $\mu(D_{S,S})$ are $\sim$0, the values of $\sigma(D_{S,S})$ are 0.8, 0.6, and 0.7 for the adder of 32-bit PE(20,0), multiplier of 32-bit PE(20,0), and 16-bit PE(20,0), respectively. An ML model trained on S is expected to generalize well for T if $D_{S,T} < \sigma(D_{S,S})$. In Table V, the values of $D_{S,T}$ consistently exceed $\sigma(D_{S,S})$ of the corresponding source distribution that was used to train the ML framework. Despite the relatively large source-to-target separation, TE_2^{cat} is almost always 0%. Thus, the ML models, trained on the criticality information of one PE, generalize well and correctly classify critical nodes

TABLE V: Evaluation of the transferability of ML1 and ML2 trained on the netlists of 32-bit PE(20,0) and 16-bit PE(20,0).

PE location	Adder in 32-bit PE						Multiplier in 32-bit PE						16-bit PE					
	F_{cr}	$D_{S,T}$	A_{ML1} (%)	A_{ML2} (%)	TE_1^{cat} (%)	TE_2^{cat} (%)	F_{cr}	$D_{S,T}$	A_{ML1} (%)	A_{ML2} (%)	TE_1^{cat} (%)	TE_2^{cat} (%)	F_{cr}	$D_{S,T}$	A_{ML1} (%)	A_{ML2} (%)	TE_1^{cat} (%)	TE_2^{cat} (%)
(45,0)	49	16.2	92	92	14.3	0	41	15.7	63.7	62.5	2.4	0	143	20.7	84.6	88.5	2.1	0
(45,8)	72	16.7	88	88	0	0	17	16	36.2	35	0	0	143	21	66.7	65.4	0.6	0
(25,16)	46	17.6	57.3	58.7	4.3	0	36	17.1	82.6	81.6	2.7	0	74	25.3	73	73	5.4	0
(21,70)	35	18.3	65.1	60.9	2.8	0	16	18.4	64.3	61.2	0	0	169	24.4	51	53	1.1	0
(16,2)	163	25.2	73.2	75.8	3	0.6	34	17.4	70.4	70.4	2.9	2.9	79	20.7	95	95	0	0
(20,0) for different workload	26	10.4	80	76.7	0	0	15	9.9	100	100	0	0	23	12.2	70	73.3	4.3	0
(45,8) for different workload	86	18.7	82	84	2.3	0	20	13.6	48	44	0	0	22	15.6	70	70	0	0

in other PEs. The results also demonstrate the effectiveness of the representative image-set used to train the 2-tier framework.

The fact that the pre-trained framework transfers to different PEs within the same array indicates that it is likely to transfer across different dataflow-based feature distributions with criticality labels based on the same application dataset on which the framework was trained. A necessary condition for the intrinsic transferability of a pre-trained framework to hold is that the end-application (e.g., MNIST, Fashion-MNIST, CIFAR-10) must remain the same because it preserves the class boundaries and label (criticality) interpretations learnt by the pre-trained ML1 and ML2. Explicit heterogeneous domain adaptation is needed to transfer an ML model from one application dataset to another application dataset [33]. Our framework is inherently transferable to PEs in the same array, as shown in Table V, due to their identical topologies. The framework needs to be retrained for the PEs to which the framework is not intrinsically transferable, i.e., PEs with very high MMD values. Moreover, our framework inherently transfers to a different set of images that belong to the same application dataset on which the framework was trained.

VIII. Conclusion

We have presented a computationally efficient framework to analyze the functional criticality of structural faults in large systolic-array AI accelerators. We have first shown that most pin-level faults for 16-bit and 32-bit PEs in a systolic array are functionally benign. We have presented a scalable DNN-based method to classify PE-internal stuck-at faults in terms of their functional criticality. We have also presented a GAN-based solution for further reducing misclassification by the DNN method. The transferability of the 2-tier classification framework has been verified for multiple PEs in different columns of the systolic array. The proposed method leads to less than 1% test escapes during the evaluation of fault criticality.

IX. Acknowledgements

This research was funded in part by the Semiconductor Research Corporation under contract no. 2879. We appreciate the contributions of Hemasagar Babu Reddy, August Ning, Jayeeta Chaudhuri, and Siddarth Madala.

Appendix

Theorem 1. *Let A represent the scenario of concurrent (multiple) s-a-0 faults on all the output stems of a set of nodes S in the PE. Let B represent the scenario of concurrent s-a-1 faults on all the output stems of the nodes in S. If both A and B are functionally benign under a criticality threshold ϵ, any combination of concurrent s-a-0 faults, s-a-1 faults, and erroneous bit-flips in the outputs of the nodes in S is also functionally benign under ϵ.*

Proof. Let the total computation (e.g., classification operation) from input (e.g., image) to output (e.g., classification output) take m iterations by the accelerator. Therefore, every node in S has a functional bit-stream containing m bits flowing through it throughout the entire duration of the computation. Let the fault-free (baseline) bit-stream flowing through a node X_i in S be represented by: $X_i^{ff} = y_1^i y_2^i..y_j^i..y_m^i$, where y_j^i is the bit present on the output of node i during j^{th} iteration of fault-free computation and $y_j^i = \{0, 1\}$. As different iterations of computation may have different contributions to the final output of the system, let each iteration j be weighted by a weight $w_j, w_j \geq 0$. When the output stems of all nodes in S are s-a-0, the faulty m-bit long bit-stream flowing through X_i is: $X_i^{sa0} = 00...0$. The net deviation d_i^{sa0} of the s-a-0 faulty bit-stream of X_i from the fault-free one is given by the weighted displacement of X_i^{sa0} from X_i^{ff}: $d_i^{sa0} = \sum_{j=1}^{m} w_j \cdot (0 - y_j^i) = -\sum_{j=1}^{m} w_j \cdot y_j^i$. As the concurrent s-a-0 faults on the output stems of nodes in S are known to be functionally benign via fault simulations, the magnitude of deviation of the individual s-a-0 bit-streams of the nodes in S from their fault-free counterparts does not exceed a certain criticality threshold, $\epsilon (\epsilon \geq 0)$. Therefore, $|d_i^{sa0}| \leq \epsilon$. Here, ϵ is determined by the criteria for functional criticality that were used to decide if the set S, when s-a-0, was functionally benign.

Similarly, when the output stems of all nodes in S are s-a-1, the faulty m-bit long bit-stream flowing through X_i is: $X_i^{sa1} = 11...1$. Hence, the net deviation d_i^{sa1} of X_i^{sa1} from X_i^{ff} is given by: $d_i^{sa1} = \sum_{j=1}^{m} w_j \cdot (1 - y_j^i)$. As the concurrent s-a-1 faults on the output stems of nodes in S are known to be functionally benign, $|d_i^{sa1}| \leq \epsilon$.

Let us now consider the scenario where one or more output stems of the nodes in S have s-a-0 faults, s-a-1 faults, and/or random bit-flips throughout the duration of m iterations. In such a scenario, an arbitrary faulty bit-stream X_i^f comprising both 0's and 1's through a node X_i in S is used to represent s-a-0 faults, s-a-1 faults, and/or random bit-flips on the output stem of X_i. Let X_i^f be represented as: $z_1^i z_2^i..z_j^i..z_m^i$ where $z_j^i = \{0, 1\}$. The net deviation d_i^f of X_i^f from X_i^{ff} is given by: $d_i^f = \sum_{j=1}^{m} w_j \cdot (z_j^i - y_j^i)$. As $z_j^i = \{0, 1\}$ and $w_j \geq 0$, if $d_i^f \leq 0$, then $|d_i^f| \leq |d_i^{sa0}| \leq \epsilon$. Similarly, if $d_i^f \geq 0$,

then $|d_i^f| \leq |d_i^{sa1}| \leq \epsilon$. This implies that the deviations of the individual bit-streams never exceed the error threshold ϵ for the given application. As a result, any combination of s-a-0/1 faults and random bit-flips on the output stems of the nodes in S are functionally benign under the criticality threshold ϵ. □

Theorem 2. *Let all combinations of stuck-at faults (single and multiple) on a set of nodes S in a PE be functionally benign. Then any internal stuck-at fault whose effect propagates only to nodes in S is also functionally benign.*

Proof. If the error due to stuck-at fault f propagates to one or more nodes in S for an input pattern, the outputs of the nodes will flip from their fault-free values. Given that one or more bit-flips for the nodes in S do not affect the functionality, the impact of error propagation due to f will be benign. □

REFERENCES

[1] V. Sze et al., "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, pp. 2295–2329, 2017.

[2] X. Guo et al., "A systolic array-based FPGA parallel architecture for the BLAST algorithm," *ISRN Bioinformatics*, vol. 2012, 2012.

[3] J. Deng et al., "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009, pp. 248–255.

[4] A. Samajdar et al., "Scale-sim: Systolic CNN accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[5] "Google Edge TPU: Coral AI". https://coral.ai.

[6] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017, pp. 1–12.

[7] W. Lu et al., "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *IEEE HPCA*, 2017, pp. 553–564.

[8] Y.-H. Chen et al., "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[9] E. Talpes et al., "Compute Solution for Tesla's Full Self Driving Computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[10] Y. Liu et al., "Deterministic Stellar BIST for Automotive ICs," *IEEE Trans. CAD*, 2019, doi:10.1109/TCAD.2019.2925353.

[11] B. Reagen et al., "Ares: A framework for quantifying the resilience of deep neural networks," in *DAC*, 2018.

[12] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *ACM SC*, 2017.

[13] J. J. Zhang et al., "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *VLSI Test Symposium*, 2018.

[14] R. Ernst et al., "Mixed criticality systems—a history of misconceptions?" *IEEE Design & Test*, vol. 33, no. 5, pp. 65–74, 2016.

[15] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] N. Jouppi et al., "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, 2018.

[17] "System Architecture of Google TPU v2/v3". https://bit.ly/2Jnurx9.

[18] H. Schütze et al., "Performance thresholding in practical text classification," in *Intl. Conf. Info. Knowledge Mgmt.*, 2006, pp. 662–671.

[19] L. Freedman et al., "The problem of underestimating the residual error variance in forward stepwise regression," *Journal of the Royal Statistical Society: Series D*, vol. 41, no. 4, pp. 405–412, 1992.

[20] P. Baldi et al., "Understanding dropout," in *NeurIPS*, 2013.

[21] A. Gebregiorgis et al., "Testing of Neuromorphic Circuits: Structural vs Functional," in *ITC*, 2019.

[22] J. J. Zhang et al., "Fault-Tolerant Systolic Array Based Accelerators for deep neural network execution," *IEEE Design & Test*, vol. 36, no. 5, pp. 44–53, 2019.

[23] Z. Xu et al., "Safety Design of a Convolutional Neural Network Accelerator with Error Localization and Correction," in *ITC*, 2019.

[24] Supplementary Document. https://bit.ly/2J8Q04m.

[25] D. Mahajan et al., "Exploring the limits of weakly supervised pretraining," in *Proc. ECCV*, 2018, pp. 181–196.

[26] T. Chen et al., "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Trans. Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.

[27] A. Radford et al., "Unsupervised representation learning with deep convolutional generative adversarial networks," *ICLR*, pp. 1–16, 2015.

[28] I. Goodfellow et al., "Generative adversarial nets," in *NeurIPS*, 2014.

[29] C. Zhang et al., *Ensemble machine learning: methods and applications*. Springer, 2012.

[30] A. Chaudhuri et al., "C-testing and efficient fault localization for AI accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[31] J. Redmon et. al., "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[32] W. Zaremba et al., "B-tests: Low variance kernel two-sample tests," *arXiv preprint arXiv:1307.1954*, 2013.

[33] O. Day et al., "A survey on heterogeneous transfer learning," *Journal of Big Data*, 2017.