# Machine Learning for Testing Machine-Learning Hardware: A Virtuous Cycle*

*(Invited Paper)*

Arjun Chaudhuri, Jonti Talukdar, and Krishnendu Chakrabarty
Department of Electrical and Computer Engineering, Duke University, Durham, NC

***Abstract*—The ubiquitous application of deep neural networks (DNN) has led to a rise in demand for AI accelerators. DNN-specific functional criticality analysis identifies faults that cause measurable and significant deviations from acceptable requirements such as the inferencing accuracy. This paper examines the problem of classifying structural faults in the processing elements (PEs) of systolic-array accelerators. We first present a two-tier machine-learning (ML) based method to assess the functional criticality of faults. While supervised learning techniques can be used to accurately estimate fault criticality, it requires a considerable amount of ground truth for model training. We therefore describe a neural-twin framework for analyzing fault criticality with a negligible amount of ground-truth data. We further describe a topological and probabilistic framework to estimate the expected number of PE's primary outputs (POs) flipping in the presence of defects and use the PO-flip count as a surrogate for determining fault criticality. We demonstrate that the combination of PO-flip count and neural twin-enabled sensitivity analysis of internal nets can be used as additional features in existing ML-based criticality classifiers.**

## I. Introduction

Deep neural net (DNN)-driven 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 [1]–[4]. Low-cost structural and functional test methods for accelerators have been proposed in [4]–[7]. Incorporation of the knowledge of fault criticality in testing enables the application of dedicated test effort for functionally critical faults. Targeted online testing of critical faults reduces the test data volume and test time required by built-in self-test solutions for complex systems. Recent work utilized supervised learning-driven DNNs, graph convolutional networks (GCNs), and neural twins to evaluate the functional criticality of stuck-at faults in the gate-level netlist of an inferencing accelerator, thereby bypassing the need for computationally expensive brute-force fault simulations [8]–[10].

The generation of labeled data for supervised learning introduces prohibitive computation costs if the labeling process involves time-consuming simulations. For criticality analysis, a large number of fault simulations are needed to collect sufficient information about critical and benign faults. In [9], nearly 20% of the fault sites in a PE were sampled for ground-truth collection via fault simulations (limited to 45 parallel simulation runs using licensed commercial software), and this process took ∼750 hours on an Intel Xeon E5 2.4 GHz CPU. Such a high runtime for collecting sufficient labeled data becomes the bottleneck in supervised learning-driven fault-criticality analysis. The amount of required labeled data for accurate classifier training is mitigated by the criticality assessment framework proposed in [10]. In [11], a probabilistic fault-grading framework was proposed for identifying potentially critical fault candidates in order to accelerate the ground-truth collection time for training criticality classifiers through targeted fault simulations. In this paper, we highlight the contributions of the criticality classifiers described in [8] and [10] in assessing the functional criticality of single stuck-at faults in a systolic array based AI accelerator. Additionally, we present the contributions of [11] to the efficient assessment of fault criticality — with and without using deep learning-enabled methods.

The remainder of the paper is organized as follows. Section II reviews systolic arrays and motivates fault-criticality analysis with limited ground-truth data. Section III presents a two-tier DNN-based framework for criticality classification. Section IV presents criticality evaluation results using the neural twin. Section IV presents the probabilistic fault-grading framework for ranking fault sites based on their functional criticality. Finally, Section VI concludes the paper.

## II. Background and Motivation

### A. Systolic Array-based Inferencing Accelerators

Systolic array-based AI accelerators allow efficient computation of multiply and accumulate (MAC) operations for inferencing workloads using state-of-the art DNNs [12]–[14]. A systolic array consists of a two-dimensional array of PEs [15]. Each PE consists of a floating-point adder and multiplier unit, and is responsible for performing a single MAC operation on the incoming stream of data. In each pass of the workload through a `4x4` systolic array, each column of the array performs the MAC operation: $S_j = \sum_{i=1}^{4} X_i \cdot \overrightarrow{W}_{j,i}$, where $S_j$ is the output of the $j^{th}$ column of the array, $X_i$ is the input activation applied to the $i^{th}$ row of the array, $W_{j,i}$ is the weight rolled into the $j^{th}$ column at the $i^{th}$ cycle. As the weight (or activation) values are rolled into the array columns, it allows overlapping the computation of multiple layers and filters concurrently in the same array, speeding up the computation significantly.

*This work was supported in part by the Semiconductor Research Corporation under GRC Task 2879.001 and Task 3106.001, and by NSF under the National AI Institute for Edge Computing Leveraging Next Generation Wireless Networks, Grant #2112562.


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

*ACM ISBN 978-1-4503-9217-4/22/10.*
*https://doi.org/10.1145/3508352.3561121*

### B. Importance of Fault-Criticality Assessment

DNNs are inherently fault-tolerant, in part due to the use of non-linear activations, regularization features, and training across large datasets [2]. In addition, their application across a wide variety of use-cases provides the end-user with flexibility in defining acceptable inferencing accuracy thresholds required for functional correctness [15]. The *functional criticality* of a structural fault is determined by its impact (degradation) on the inferencing accuracy for a given target application and a given test set $T$. A fault is deemed to be *functionally critical* if the inferencing accuracy in its presence degrades below a pre-determined accuracy threshold, $A_{min}$, which is evaluated based on the safety criticality of the target application [15]. Similarly, a fault is considered to be *functionally benign* if it does not reduce the inferencing accuracy beyond $A_{min}$.

Analyzing the functional criticality of structural faults can be useful for guiding test-effort, reducing yield loss due to functionally benign manufacturing defects, and performing the quality assessment of AI accelerators throughout their entire life-cycle [7]. As the functional criticality of structural faults depends on the type of application/dataset, the architecture of the hardware accelerator, mapping of the application DNN to the target hardware, and the pre-determined criticality threshold, the same fault-criticality assessment methodology can be utilized to evaluate and catalog fault-criticality across different domain-specific use-cases.

### C. Need for Selective Fault Simulation

Data reuse in systolic array-based AI accelerators implies that a single fault in the PE can affect multiple neurons in the mapped DNN model. Thus, identifying the functional criticality of structural faults requires numerous fault simulations, sometimes across several inputs (for e.g., several images in the test dataset) for a given target application. Prior work has focused on utilizing supervised learning methods such as DNNs [8], GCNs (graph-convolutional networks) [9], and neural twins [10] for functional fault criticality classification. These models perform better when supplied with large amounts of labeled data for training. However, given that a single 32-bit PE can contain thousands of stuck-at faults, the total number of structural faults in an industry scale systolic array ($128 \times 128$ or $256 \times 256$) can be in the range of millions [8]. As shown in [9], it can take up to 750 hours to simulate only $20\%$ of the faults in a single PE with parallelization factor of 45:1 (often dictated by commercial licensing constraints). In addition, the percentage of critical faults in a systolic array is much less than the percentage of benign faults. Over $85\%$ of structural faults in the MAC units of a 32-bit PE (both adder and multiplier units) are potentially benign, while close to $80\%$ of structural faults in the MAC units of the 16-bit PE are potentially benign [8]. The lack of informed methods that are designed to select candidate fault sites for fault simulation exacerbates the problem of identifying a sufficient number of critical fault sites while minimizing the number of fault simulations.

## III. Criticality Assessment using Deep Learning

### A. Two-Tier DNN Framework for Reducing Misclassifications

In [8], a two-tier DNN-based framework is used to classify fault criticality with a low test escape; see Fig. 1. 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 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. The above set of nodes is randomly partitioned into training and validation datasets in the ratio 3:1. 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 $S_{TE}$. The set $S_{TE}$ is used for training a generative adversarial network (GAN) to generate samples with features matching the feature distribution of the test-escape nodes. 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 correctly classified as benign by ML1, is used to train ML2, to distinguish between real and misclassified benign nodes.

### B. Criticality Evaluation

During the evaluation phase of aggregated GAN-based criticality assessment flow, fault simulation of nodes, not present

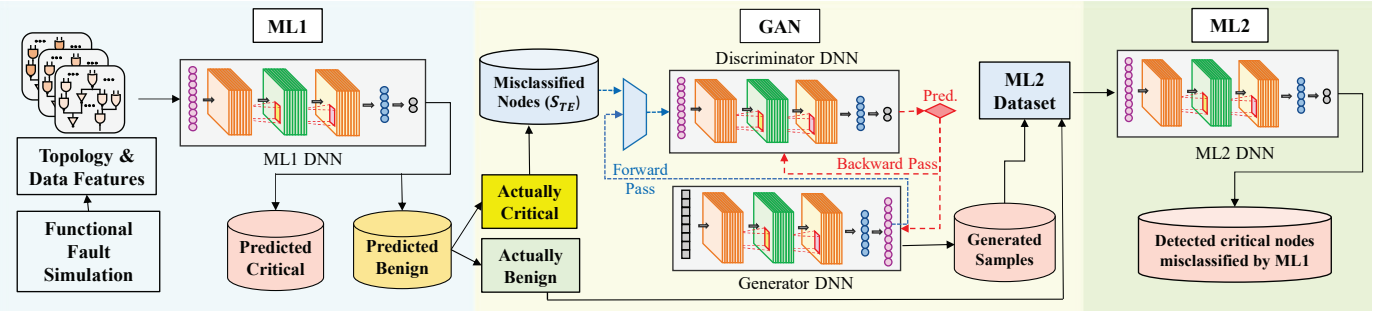


Fig. 1: Methodology used for training aggregated GAN-based criticality assessment flow.

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. We achieve an average test-escape of less than 3% and 1% for the criticality assessment of the 16-bit and 32-bit PEs, respectively.

## IV. Criticality Analysis using a Neural Twin

In [10], the standard cell instances in the PE netlist are substitued with their corresponding Cell-Nets while translating PE into PE-Net. A multi-layer perceptron (MLP) architecture is used to build a Cell-Net. The Cell-Net is then trained based on the truth table of the corresponding logic gate. During training, the Cell-Net model parameters are regressed in order to estimate the Boolean output of the logic gate for a given set of Boolean (or binary) inputs.

### A. Modeling of Cell-Net using MLP

A Cell-Net is mathematically represented by a non-linear function, $\phi_{cell}$, and is composed of an MLP configuration with three fully connected layers: $\phi_{cell}(\mathbf{z}_{in}) = \sigma_{sig}(A_3\sigma_{sig}(A_2\sigma_{sig}(A_1\mathbf{z}_{in} + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3)$. Here, $\mathbf{z}_{in}$ denotes the binary input vector of size $r_z$; $A_1$, $A_2$, and $A_3$ are affine transformations of size $4 \times r_z$, $4 \times 4$, and $1 \times 4$ respectively; $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{b}_3$ are bias vectors of the MLP network. The Sigmoid activation function present at the output of every neuron in the Cell-Net is denoted by $\sigma_{sig}(\cdot)$. The input vector $\mathbf{z}_{in}$ is composed of the binary inputs to the standard cell. When $z_1$ and $z_2$ are binary inputs to a standard cell, $\mathbf{z}_{in} = [z_1, z_2]$. The size of the input vector, $r_z$, is equal to the number of input ports of the standard cell being modeled. For example, when the cell being modeled is an AND2 gate, $r_z$ is 2. The final layer of the Cell-Net contains a single neuron that outputs the binary state of the gate being modeled. The Sigmoid activation function constrains the Cell-Net's output to be between 0 and 1 as the objective of Cell-Net modeling is to estimate the binary output of a logic gate. The Cell-Net modeling is illustrated in Fig. 2(A).

### B. Construction of a Neural Twin

We convert the PE netlist into a directed acyclic netlist-graph $G$ to build the PE-Net, i.e., the neural twin of the PE. The conversion maps each cell instance in the netlist to its corresponding Cell-Net, which is represented by a vertex in $G$. For instance, a 2-input AND gate (AND2) is mapped to the AND-Net with two inputs and one output. A wire connecting two cell instances in the netlist is mapped to a neural connection between the corresponding Cell-Nets, i.e., an edge between the corresponding vertices in $G$. The neural-twin modeling is illustrated in Fig. 2(B).

The objective of PE-Net is to produce the partial-sum output, $y^{ps}_{r,c}$, of PE$(r, c)$, i.e., the PE in the $r$-th row and $c$-th column (0-indexed) of the systolic array. In other words, the PE-Net is a neural network-based representation of a PE in the systolic array. The PE-Net is represented in functionality by: $y^{ps}_{r,c} = w \cdot x + y^{ps}_{r-1,c}$, where $x$ and $w$ are $n$-bit floating-point (FP) scalars that interchangeably propagate activation and weight signals to a PE across different inferencing cycles; $n$ is 16 or 32 depending on the PE architecture. The $n$-bit FP $y^{ps}_{r-1,c}$ carries the partial-sum output of PE$(r-1, c)$, becoming the partial-sum input to PE$(r, c)$. The PE-Net architecture is based on the topology of $G$ such that there exists a one-to-one physical correspondence between each wire (or fault site) in the PE netlist and a neural connection in the PE-Net. Every logic gate in the PE netlist is substituted by the corresponding Cell-Net in the PE-Net.

Let the neural connection $e_{ij}$ connect Cell-Nets $v_i$ and $v_j$ in the PE-Net. We associate a bias term $bias_{ij}$ with every $e_{ij}$. The bias term represents a perturbation in the signal being propagated along $e_{ij}$. Therefore, the output $e_{ij}$ of $v_i$ is added to $bias_{ij}$ and the resultant sum $s_{ij}$ is passed to the input of $v_j$. To constrain $s_{ij}$ to lie between 0 and 1, we pass $s_{ij}$ through an activation function, $clp(\cdot)$: $clp(z) = \max\{\min\{0, z\}, 1\}$, where $z \in \mathbb{R}$. Let the $n$ partial-sum output bits of PE-Net be denoted by $\{z_{out,0}, z_{out,1}, ..., z_{out,n-1}\}$, with $z_{out,0}$ and $z_{out,n-1}$ being the least (LSB) and most significant (MSB) bits, respectively. Here $n$ can be 16 or 32 depending on the PE architecture. The $n$ output bits are converted into a single FP scalar value, $\hat{y}^{ps}_{r,c}$, using the following differentiable expression: $\hat{y}^{ps}_{r,c} = (1-2z_{out,n-1}) \cdot \left(2^{\left(\sum_{i=0}^{n_e-1}(2^i \cdot z_{out,n-n_e+i-1}) - 2^{n_e-1}+1\right)}\right) \cdot \left(1 + \sum_{i=1}^{n_m}(2^{-i} \cdot z_{out,n-n_e-i-1})\right)$. Here, $n_e$ and $n_m$ denote the number of exponent and mantissa bits in the $n$-bit binary representation of a FP value whose MSB represents sign bit.

To enable backpropagation in the PE-Net, we ensure that the network is end-to-end differentiable by including pseudo-differentiable activation functions.

### C. Criticality Classification using MDT and Decision Tree

The misclassification-driven training (MDT) algorithm [16] used to identify critical faults (CFs) in PE-Net. Based on the bias sensitivity computed by MDT, we introduce a selective bias (+1 or -1) to maximize the error of the neural-twin MLP model with respect to the fault-free PE. Then, we train a decision tree (DT) model using a limited amount of ground truth to classify the neural-twin's output error as functionally critical or benign. Fig. 2(C-D) illustrate the MDT-based methodology for criticality analysis.

MDT iterates through fault locations and introduces a bias at the location $l$ to simulate a stuck-at fault ($F$) effect. We update the bias at $l$ according to the sign (direction) of the gradient; the change in direction of the bias is the same as the gradient at $l$, or $\nabla[l]$. This can be viewed as a gradient *ascent* step to maximize the error function. We next obtain the error for $F$ by inferencing using the updated neural twin. We use the pre-trained DT model to predict fault criticality based on the obtained error value. Almost all faults identified using MDT lead to extremely large MSE and many of these faults are predicted as being critical by the DT.

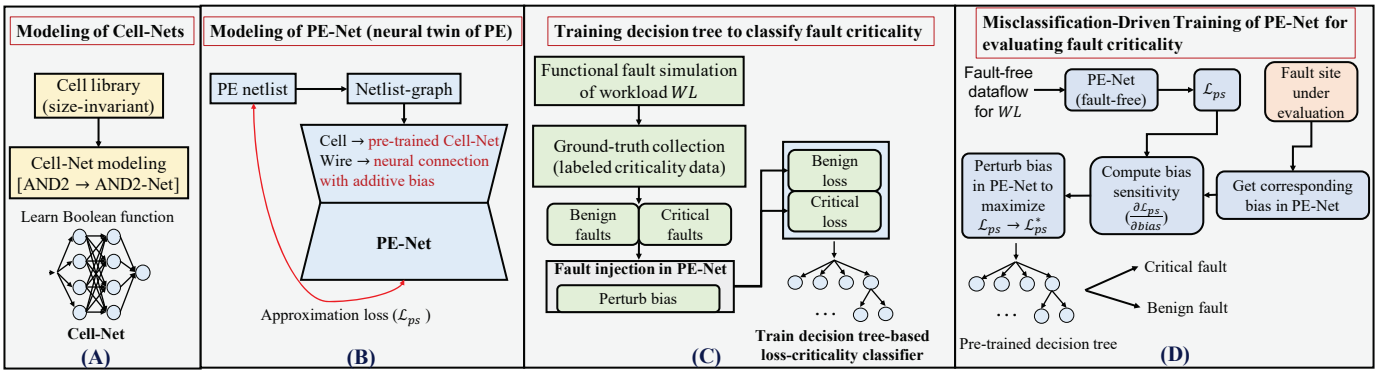


Fig. 2: Functional criticality analysis of structural faults using the neural twin of a PE.

TABLE I: Criticality evaluation using neural twin.

| $f$ (%) | 16-bit PE | | 32-bit PE | |
|---|---|---|---|---|
| | $A_c$ (%) | $M_c$ (%) | $A_c$ (%) | $M_c$ (%) |
| 2 | 97.4 (5.3) | 5.1 (2.1) | 94.3 (1.07) | 13.3 (1.3) |
| 5 | 100 (0) | 0 (0) | 98.9 (0.3) | 3.5 (0.7) |
| 8 | 100 (0) | 0 (0) | 100 (0) | 0 (0) |
| 10 | 100 (0) | 0 (0) | 100 (0) | 0 (0) |

### D. Evaluation Results

Table I presents the evaluation results for the 16-bit and 32-bit PE(20,0) for different values of $f$ and by using different criticality analysis frameworks; $f$ is the percentage of ground-truth data used to train DT. The percentage classification accuracy is denoted by $A_c$ and the percentage of critical fault locations misclassified as benign is denoted by $M_c$. The neural twin-based framework achieves $A_c = 100\%$ and $M_c = 0\%$ misclassification of critical faults by using the DT trained on only 8% of the labeled data for the 32-bit PE and 5% of the labeled data for the 16-bit PE.

## V. Probabilistic Fault Grading for Selective Fault Simulation

### A. Fault-Criticality Grading

#### 1) Computing Probability of Transition Fault

As shown in [11], consider the probability distribution function (PDF) of the additive bias $b_x$ that affects the signal being propagated by the output net ($n_x$) of a defective cell instance in the PE netlist. This additive bias is capable of modeling a cell-internal resistive (or delay) defect, temporal fluctuations in the power supply due to IR drop or voltage droop, and thermal noise-induced perturbations in the gate's output voltage level. The sign and magnitude of $b_x$ determine the extent to which the signal carried by the faulty net is affected. In case of cell-internal delay defects, the defect size is modeled by the bias magnitude and the defect location (PUN or PDN) can be mapped to the bias sign. Moreover, the noise margin of the digital circuit is modeled using a non-negative tolerance parameter, $\tau_b$. If the voltage level of VDD (ground) is mapped to '1' ('0') and the rule-of-thumb noise margin is 20% of VDD, then $\tau_b = 0.2$. The additive bias is likely to cause a faulty signal transition (i.e., bit flip) if its magnitude exceeds $\tau_b$. By considering the PDF, $p(b_x)$, of $b_x$ on a faulty net and the circuit's intrinsic tolerance to faults (i.e., noise margin), we account for the fact that not all defects or non-idealities are critical to circuit functionality; the impact on functionality depends on the location, size, and likelihood of defect occurrence. We next compute the probability of a failed signal transition (0→1 or 1→0) on the faulty net ($n_x$) using the bias distribution function. While we consider a single fault in the PE netlist for criticality analysis, the proposed fault-grading framework can be extended to multiple-fault scenarios.

Without loss of generality, we assume a PDF for the bias that is symmetric around the y-axis, i.e., $p(b_x) = p(-b_x)$. This implies that the likelihood of occurrence of a certain bias size is the same irrespective of the bias sign. In the context of delay defects, the defect-size distribution is assumed to stay the same for defects in PUN and PDN. Let the probability of signal '1' being propagated by fault-free $n_x$ be $p_1$. Let the probability of signal '0' being propagated by fault-free $n_x$ is $p_0$. Let $E_{ff,0}$ denote the event that $n_x$ carries '0' in the fault-free scenario. Let $E_{flt,0}$ denote the event that $n_x$ does not undergo a 0→1 transition in the presence of the fault, i.e., $b_x$. Therefore, $P[E_{ff,0} \cap E_{flt,0}]$ denotes the probability that the fault-free signal is '0' and remains '0' under faulty conditions. Using Bayes' theorem, $P_0 = P[E_{ff,0} \cap E_{flt,0}] = P[E_{flt,0}|E_{ff,0}] \cdot P[E_{ff,0}]$. Note that $P[E_{ff,0}] = p_0$.

The logic value implied by the faulty signal equals that of the fault-free signal ('0') under two conditions: (i) $b_x \leq 0$: the bias-added signal becomes negative and is correctly perceived as logic '0' by the downstream logic gates; (ii) $0 < b_x < \tau_b$: the signal value on $n_x$ becomes $b_x$ which is within the circuit noise margin and does not cause a logical 0→1 transition, i.e., no 1→0 transition fault. The probability that one or both of the above conditions hold equals $P[E_{flt,0}|E_{ff,0}]$. The probability of $b_x \leq 0$ is computed as area under the PDF curve between $b_x = -\infty$ and $b_x = 0$: $\int_{b_x=-\infty}^{0} p(b_x) \mathrm{d}b_x$. The probability of $0 < b_x < \tau_b$ is computed as the area under the PDF between $b_x = 0$ and $b_x = \tau_b$, which is $\int_{b_x=0}^{\tau_b} p(b_x)\, \mathrm{d}b_x$. Therefore, $P_0 = p_0 \cdot \int_{b_x=-\infty}^{\tau_b} p(b_x)\, \mathrm{d}b_x$.

Similarly, the logic value implied by the faulty signal equals that of the fault-free '1' (i.e., VDD or power supply) under two conditions: (i) $b_x \geq 0$: the bias-added signal exceeds 1 and is correctly perceived as logic '1' by the downstream logic gates; (ii) $-\tau_b < b_x < 0$: the signal value on $n_x$ becomes $1 - b_x$ which is within the circuit noise margin and does not cause a logical 1→0 transition, i.e., no 0→1

transition fault. The probability that the faulty signal carried by $n_x$ implies logic '1', given that the fault-free signal is also '1', is given by: $\int_{b_x=-\tau_b}^{\infty} p(b_x)\,\mathrm{d}b_x$. Therefore, the probability that the fault-affected $n_x$ propagates logic '1' is given by: $P_1 = p_1 \cdot \int_{b_x=-\tau_b}^{\infty} p(b_x)\,\mathrm{d}b_x$. Consequently, the probability ($P_{flip}$) that $n_x$ undergoes a logical transition in the presence of a fault is: $P_{flip} = 1 - P_0 - P_1$.

The defect-tolerance probability, $P_{tol}$, is defined as the area under the PDF between $b_x = -\tau_b$ and $b_x = \tau_b$. In this region, the additive bias does not affect the logic switching of the faulty net. A popular choice for the defect-size distribution function, $h(x)$, is $h(x) = a \cdot e^{-b\cdot|x|}$, where $x$ is the defect size, and $\{a, b\}$ are fitting parameters. We adopt the same function for $p(b_x)$. As $a \cdot e^{-b\cdot|b_x|}$ is symmetric around y-axis and $\int_{-\infty}^{\infty} p(b_x)\,\mathrm{d}b_x = 1$, $\int_0^{\infty} a \cdot e^{-b\cdot|b_x|}\,\mathrm{d}b_x = 0.5$. This implies $b = 2a$. Therefore, $P_{tol} = \int_{-\tau_b}^{\tau_b} p(b_x)\,\mathrm{d}b_x = b\int_0^{\tau_b} e^{-b\cdot b_x}\,\mathrm{d}b_x = 1 - e^{-b\cdot\tau_b}$. Given $\tau_b$ and $b$, we can compute $P_{tol}$. Moreover, we can express $P_{k=0,1}$ as $P_k = p_k \cdot (\frac{1+P_{tol}}{2})$.

### 2) Propagation of Transition-Fault Probabilities

We convert the pipelined PE netlist into a directed acyclic graph $G$ where a vertex or node represents a cell instance and an edge represents a wire connection between two cell instances. The primary input (PI) and primary output (PO) pins are mapped to graph nodes as well. We traverse $G$ in a topologically sorted order. The $P_V$ is computed for every node's output stem during the traversal. The probability tuples for the PE's primary inputs (PIs) are extracted from the distribution of 1's and 0's appearing at the PIs across all inferencing cycles of the applied workload. By considering the bit distribution of the input workload, the notion of workload-dependent circuit functionality is incorporated into the computed transition-fault probabilities. In this work, we consider a workload of 100 MNIST images being classified by the LeNet-5 model mapped to the systolic array. Each image requires 1298 cycles (or iterations of PE reuse) to produce the final inferencing result. Therefore, each PE receives bitstream corresponding to a total of $1298 \times 100$ cycles. For each PI node in $G$, the fault-free $p_0$ ($p_1$) is calculated as the fraction of 0s (1s) appearing in the 129800-cycle fault-free bitstream.

During topological traversal of $G$, the $P_V$ tuple computed for a parent node (driver gate) is used to calculate fault-free $p_k (k \in \{0, 1\})$ for the child node (load gate) based on the controlling/non-controlling Boolean values of the child. From the computed $p_k$, $P_k$ of the child is calculated as $P_k = p_k \cdot (\frac{1+P_{tol}}{2})$, if the child node contains a fault with $P_{tol} < 1$; if child is fault-free, $P_{tol} = 1$ implying $P_k = p_k$.

The $P_V$ computation procedure exits once $P_V$ tuples have been computed for all PO nodes in $G$. Separate $P_V$ tuples are calculated for the different output ports of a multi-output logic gate, e.g., SUM and CARRY-OUT ports of a full-adder (FA). The nodes corresponding to complex gates such as FAs, OAIs, and AOIs are decomposed into primitive gates (e.g., 2-input AND, OR, XOR, INV, BUF, etc.) prior to $P_V$ computation.

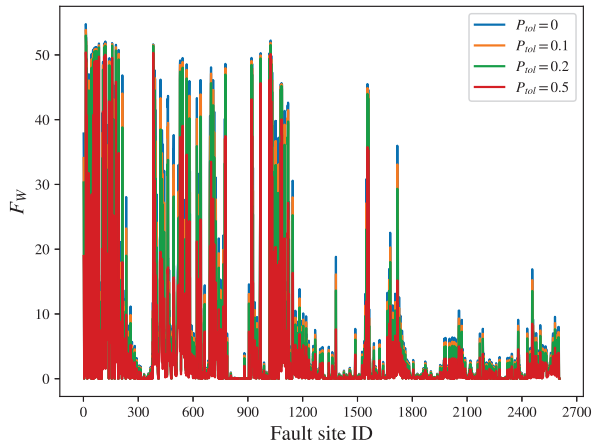


Fig. 3: Variation of $F_W$ with variation in $P_{tol}$.

### 3) Ranking Fault Sites using Transition-Fault Probabilities

After the $P_V$ tuples are calculated for the PO nodes in the PE netlist in the presence of a fault $f$, the expected number of POs flipping ($F_E$) under faulty conditions is calculated by summing the $P_{flip}$ probabilities of all PO nodes corresponding to the partial-sum output ports of the PE. For $m$ POs, the expected bit-flip count $F_E$ is given by $F_E = \sum_{i=1}^{m} P_{flip,PO_i}$, where $P_{flip,PO_i}$ is the $P_{flip}$ value in the $P_V$ computed for the $i$-th PO, $PO_i$. A higher value of $F_E$ indicates a higher likelihood of $f$ being functionally critical.

While $F_E$ allots equal weight to all POs of the PE, in reality, faulty transitions in some POs may have greater functional impact on the final inferencing accuracy compared to other PEs. For the floating-point PE's partial-sum output bus, a flip in the most-significant bit (MSB) is likely to be more critical than a flip in the least-significant bit (LSB). We compute a weighted bit-flip count, $F_W$, by assigning larger weights to the more significant bits of the output bus: $F_W = \sum_{i=1}^{m} i \cdot P_{flip,PO_i}$. Here, $PO_m$ ($PO_1$) is the MSB (LSB). Fig. 3 shows the increase in $F_W$ with decrease in $P_{tol}$ in 16-bit PE(20,0).

We also compute a functionality-aware weighting of the PO bit-flips. The combination of binary partial-sum outputs of the floating-point PE is interpreted as a floating-point operand when the DNN model's parameters are mapped to the systolic array. The impact of a fault on the inferencing accuracy will depend on the functional nature of the flipping POs, i.e, the POs where the fault effect propagates. Therefore, we assign weights to the different POs depending on whether they represent the sign, exponent, or mantissa bits of the partial-sum bus. The weight assigned to a PO indicates the maximum magnitude of relative error in the partial-sum value in decimal representation when the PO flips. Consider a fault-free partial-sum value of $M \times 2^E$, where $M$ ($E$) denotes the decimal representation of the mantissa (exponent bits); here $1 \leq M < 2$. Let there be $n_e$ ($n_m$) exponent (mantissa) bits. If the sign bit flips, the partial-sum value changes to $-M \times 2^E$; the resulting relative error is $\frac{2M\times 2^E}{M\times 2^E} = 2$. If the $i$-th exponent bit flips ($i = n_e - 1$ is the MSB of the exponent bits, $i = 0$ being the LSB), the partial-sum value changes to $M \times 2^{E+2^i}$.

The resulting relative error is $\frac{M\times 2^E\cdot(2^{2^i}-1)}{M\times 2^E} = 2^{2^i}-1$. If the $j$-th mantissa bit flips ($j = n_m$ is the LSB of the mantissa bits, $j = 1$ being the MSB), the partial-sum value changes to $(M + 2^{-j}) \times 2^E$. The resulting relative error is $\frac{2^{-j}\times 2^E}{M\times 2^E} = \frac{2^{-j}}{M} \leq 2^{-j}$. Thus, the functionality-aware weighted PO-flip count, $F_{FW}$, is given by: $F_{FW} = 2 \cdot P_{flip,PO_{\text{sign}}} + \sum_{i\in\text{exponent}}(2^{2^i}-1)\cdot P_{flip,PO_i} + \sum_{j\in\text{mantissa}} 2^{-j}\cdot P_{flip,PO_j}$.

### B. Sensitivity Analysis using Neural Twin

Let $x_i$, $w_i$, $y^{ps}_{r-1,c,i-1}$, and $y^{ps}_{r,c,i}$ denote $n$-bit activation input, $n$-bit weight input, $n$-bit partial-sum input, and floating-point partial-sum output of a PE in the $i$-th inferencing cycle for an input image, respectively. Let the number of MNIST images in the application workload be $m$. We use our in-house Python-based framework to carry out fault-free inferencing for the $m$ images and collect the fault-free cycle-wise dataflow $\{x_i, w_i, y_{ps,in}, y_{ps}; 1 \leq i \leq m\}$ through the PE. If the $j$-th image ($I_j$) in the workload requires $t$ inferencing cycles through the systolic array to produce the final classification, our dataflow matrix $D_j$ consists of $t$ rows and $3n+1$ columns. The first $3n$ columns in $D_j$ contain the binary inputs to the PE and the final column contains the partial-sum output of the PE, $y_{ps}$, which acts as the label for training PE-Net.

During training of the PE-Net, the input feature vectors corresponding to the $t$ inferencing cycles of an image are passed to the PE-Net. Next, we compute the mean-squared-error (MSE) loss, $L_{ps}$, between $y_{ps}$ and $\hat{y}_{ps}$ for the $t$ input vectors. During the training of PE-Net, only the bias parameters of the PE-Net are updated. We report the mean-absolute-error (MAE) loss for the PE-Net output while using the MSE loss for backpropagation. The batch size, denoted by $bs$, indicates the number of images across which the $L_{ps}$ is accumulated. The accumulated $L_{ps}$ is then averaged and used for backpropagation to compute the loss gradients with respect to the PE-Net's biases using PyTorch autograd. The gradient of $L_{ps}$ with respect to a bias $bias_i$ is given by $\frac{\partial L_{ps}}{\partial bias_i}$; it measures the sensitivity of the PE output to a small change in $bias_i$. The computed gradients are used to update the biases using gradient-descent policy for minimizing $L_{ps}$. One training epoch ends when the PE-Net has encountered all $m$ images in the input dataset. The average $|\frac{\partial L_{ps}}{\partial bias_i}|$ across all training epochs is a direct estimate of the significance of the signal carried by the net $n_i$, corresponding to $bias_i$, in determining the PE output. Thus, larger $|\frac{\partial L_{ps}}{\partial bias_i}|$ implies higher likelihood of a fault in $n_i$ being critical [11].

### C. Towards Criticality Classifiers with Augmented Features

We train the PE-Net on workload-derived bitstream samples of the PE to obtain the gradients of the internal biases [11]. The gradient of PE-Net's output error ($L_{ps}$) with respect to a bias ($bias_i$) is given by $\nabla L_{ps,bias_i} = \frac{\partial L_{ps}}{\partial bias_i}$. After obtaining $F_{FW}$ and $|\nabla L_{ps}|$ (averaged across training epochs) for a fault site and its corresponding bias in a source PE, we use them as features to train a Random-Forest model, having a pareto-optimal tree depth of 30, for predicting inferencing accuracy in the presence of the fault. The model trained on the ground-truth accuracies and features of the source PE is transferred to predict fault-induced inferencing accuracies for identifying critical faults in a target PE. Table II presents the results of the model transfer between two 16-bit PEs; $A_{th} = 90\%$. The rightmost column presents the $R^2$ scores of the models trained on the respective source PEs. High scores of 0.82 and above indicate that the model is able to derive a meaningful non-linear relationship between the inferencing accuracy and the feature combination of $F_{FW}$ and $\nabla L_{ps}$. This implies that $F_{FW}$ and $\nabla L_{ps}$ are reliable machine-learnable features that can be used to augment the feature set of existing criticality classifiers and improve their classification accuracy.

TABLE II: Transfer of Random-Forest model using $|\nabla L_{ps}|$ and $F_{FW}$ as features for criticality analysis.

| SourceTarget | (20,0) | (25,16) | (45,8) | (21,70) | $R^2$ |
|---|---|---|---|---|---|
| (20,0) | - | 25 (41) | 27 (40) | 8 (23) | 0.86 |
| (25,16) | 219 (307) | - | 31 (40) | 21 (23) | 0.88 |
| (45,8) | 292 (307) | 32 (41) | - | 20 (23) | 0.91 |
| (21,70) | 247 (307) | 25 (41) | 30 (40) | - | 0.82 |

## VI. Conclusion

The two-tier DNN-based framework classifies critical faults with a negligible number of misclassifications. In the neural twin-based framework, the MDT selectively injects a fault in the neural twin of the PE based on bias sensitivity and leverages a DT to classify the PE's output error as benign or critical. Neural twin-enabled bias gradient and the weighted bit-flip metric have been shown to be useful features for training criticality classifiers.

## References


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

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

[3] 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.

[4] S. Kundu et al., "Toward functional safety of systolic array-based deep learning hardware accelerators," *TVLSI*, 2021.

[5] A. Chaudhuri et al., "C-testing of AI accelerators," in *ATS*, 2020.

[6] A. Gebregiorgis et al., "Testing of neuromorphic circuits: Structural vs functional," in *ITC*, 2019.

[7] M. Sadi et al., "Test and yield loss reduction of AI and deep learning accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[8] A. Chaudhuri *et al.*, "Functional criticality analysis of structural faults in ai accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.

[9] A. Chaudhuri et al., "Fault-criticality assessment for AI accelerators using graph convolutional networks," in *DATE*, 2021.

[10] A. Chaudhuri *et al.*, "Efficient fault-criticality analysis for AI accelerators using a neural twin," in *2021 IEEE International Test Conference (ITC)*, 2021.

[11] A. Chaudhuri et al., "Probabilistic fault grading for AI accelerators using neural twins," in *ISVLSI*, 2022.

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

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

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

[15] A. Chaudhuri et al., "Functional criticality classification of structural faults in AI accelerators," in *ITC*, 2020.

[16] C. Chen et al., "Efficient identification of critical faults in memristor crossbars for deep neural networks," in *DATE*, 2021.