STRIVE: Empowering a Low Power Tensor Processing Unit with Fault Detection and Error Resilience

NOEL DANIEL GUNDI, Electrical and Computer Engineering, Utah State University, College of Engineering, Logan, United States
SANGHAMITRA ROY, Electrical and Computer Engineering, Utah State University, Logan, United States
KOUSHIK CHAKRABORTY, Electrical and Computer Engineering, Utah State University, Logan, United States

Rapid growth in Deep Neural Network (DNN) workloads has increased the energy footprint of the Artificial Intelligence (AI) computing realm. For optimum energy efficiency, we propose operating a DNN hardware in the Low-Power Computing (LPC) region. However, operating at LPC causes increased delay sensitivity to Process Variation (PV). Delay faults are an intriguing consequence of PV. In this article, we demonstrate the vulnerability of DNNs to delay variations, substantially lowering the prediction accuracy. To overcome delay faults, we present STRIVE—a post-fabrication fault detection and reactive error reduction technique. We also introduce a time-borrow correction technique to ensure error-free DNN computation.

CCS Concepts: • **Hardware → Very large scale integration design**; *Tensor Processing Unit;*

Additional Key Words and Phrases: Tensor processing unit (TPU), deep neural network (DNN), low-power computing (LPC), process variation (PV), fault detection, timing error resilience

ACM Reference Format:
Noel Daniel Gundi, Sanghamitra Roy, and Koushik Chakraborty. 2025. STRIVE: Empowering a Low Power Tensor Processing Unit with Fault Detection and Error Resilience. *ACM Trans. Des. Autom. Electron. Syst.* 30, 2, Article 16 (January 2025), 25 pages. https://doi.org/10.1145/3705003

1 Introduction

The emergence of **Deep Neural Networks** (**DNNs**) and their growing usage in diverse applications has led to a rapid advancement in the development of **Application Specific Integrated Circuit** (**ASIC**) architectures for the **Artificial Intelligence** (**AI**) computing realm. A **Tensor Processing Unit** (**TPU**) is one such ASIC, developed by Google and has been deployed in their datacenters for the inference phase of the DNN computation [20]. With the rapid deployment of AI hardware at the edge, there is a tremendous need to investigate these circuit-architectures in the **Low-Power Computing** (**LPC**) region [4].

Figure 1 demonstrates a key challenge in realizing AI hardware at the edge: a massive delay variance with voltage at LPC in comparison to **Super Threshold Computing** (**STC**). The figure

Author's Contact Information: Noel Daniel Gundi, Electrical and Computer Engineering, Utah State University, College of Engineering, Logan, Utah, United States; e-mail: noeldaniel.gundi@usu.edu; Sanghamitra Roy, Electrical and Computer Engineering, Utah State University, Logan, Utah, United States; e-mail: sanghamitra.roy@usu.edu; Koushik Chakraborty, Electrical and Computer Engineering, Utah State University, Logan, Utah, United States; e-mail: koushik.chakraborty@usu.edu.

ACM 1084-4309/2025/01-ART16
https://doi.org/10.1145/3705003

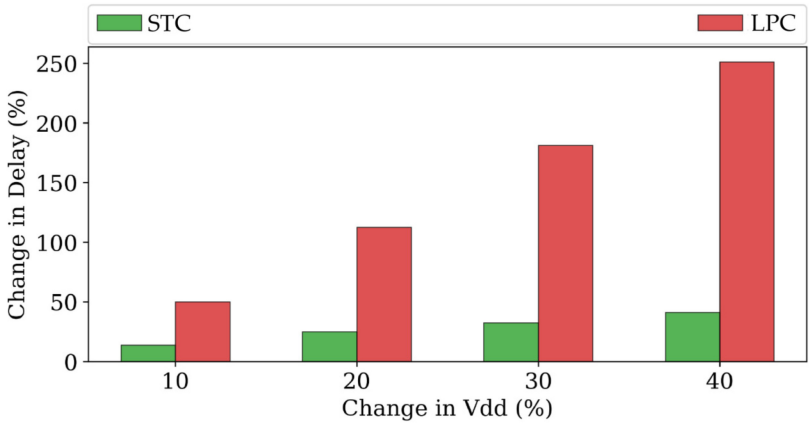


Fig. 1. Delay sensitivity for power supply at STC vs LPC. HSPICE simulations shown for a 31 fan-out-of-four (FO4) inverter chain at the 14 nm multi-gate technology node [54].

shows a FO4 inverter delay characteristics at these two regions, when the supply voltage is varied by the same percentage variation of the respective voltage domains. For example, a 40% voltage variation can cause a 45% delay variation at STC, and a huge 250% delay variation at LPC. Due to these extreme sensitivities, the omnipresent manufacturing **Process Variation** (**PV**) in the devices can realize gate delays up to 20× of the nominal values in the LPC region [42]. Collectively, a small set of PV-affected gates, in combination with a minute variation in operating condition (e.g., voltage), can completely alter the critical path of the circuit and protract the combinational delay, which subsequently leads to frequent timing violations. We term these delay faults as **Low-Power Faults** (**LP-faults**) in this article.

LP-faults may remain benign and exhibit the correct operation at nominal voltages. However, they are exposed and cause frequent timing faults at LPC. The process of frequency guard-banding which works efficiently at STC, becomes highly ineffective at LPC due to the extreme variation in delays. In a tightly pipelined architecture such as TPU with a large number of interconnected **Multiplier-and-Accumulate** (**MAC**) units, detection of an LP-fault in an individual MAC unit can be a formidable challenge. Furthermore, as these faults manifest only after fabrication and at certain operating conditions, detecting and correcting these faults become even more challenging. In this article, we analyze the damage an LP-fault can induce on a DNN inference operation.

Interestingly, many modern datasets used in DNN computations exhibit a plethora of zero weights [7, 39], as well as, zero activation elements. In conjunction with the perceived resilience of DNN software from errors, it is intuitive to expect that DNN inference may be inherently tolerant to LP-faults [19]. However, our rigorous cross-layer analysis reveals otherwise. For example, an otherwise innocuous zero result from a MAC can lead to non-zero outcome under an LP-fault, causing a havoc in the inference accuracy. Not only do we find that inference accuracy can drop dramatically from LP-faults, our results show extreme unpredictability in these inference drops, demonstrating a critical need to rigorously analyze and mitigate LP-faults for successful deployment of these AI hardware accelerators at the edge.

Razor is a popular technique which uses a double sampling flip-flop to detect a timing error in a pipelined circuit and employs an instruction replay to recompute the erroneous data [14]. However, replaying an instruction in a TPU pipeline requires stalling of the entire systolic array operation,

incurring a massive loss in throughput for such a data-parallel architecture. *To our knowledge, our paper is the first work to introduce a post-fabrication fault detection technique to identify the faulty MAC units in a TPU affected by LP-faults.*

Our specific contributions in this article are as follows:

— We explore the impact of delay faults in a systolic array of MAC units. We investigate the problem an LP-fault can pose in transforming a zero output from a multiplier to an incorrect non-zero value (Sections 2.2 and 2.3). We also show how a subset of zero computations in a DNN matrix multiplication magnifies this threat (Section 2.3).
— We introduce STRIVE—a low-overhead faulty MAC detection technique for a TPU systolic array (Section 3.5), to identify the PV-affected MAC units and timing error resilience techniques to mitigate the effect of LP-faults (Sections 3.4 and 3.6.3).
— We demonstrate that STRIVE schemes incurs less than 1% loss in inference accuracy for 8 DNN benchmarks in a TPU affected by a gate level fault rate of 1% (Section 5.2). Additionally, STRIVE techniques give 1.8×, 1.2×, and 2.0× better performance per unit power than Fault-Aware Pruning [53].

2 Motivation

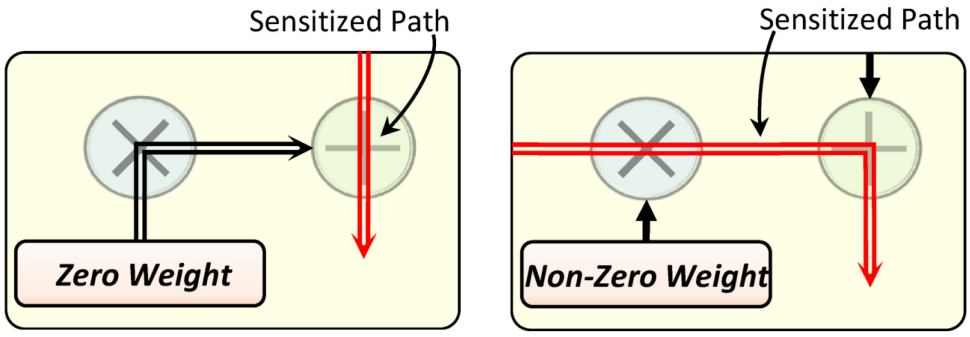
In this section, we will uncover a post fabrication phenomenon which poses a severe threat to the error resilience of DNNs. Moreover, we also demonstrate the threat posed by a multiplier unit, when an expected zero computational output results in an unpredictable non-zero value. Sections 2.1 and 2.2 provide a background of the TPU and LP-faults, respectively. Sections 2.3 and 2.4 elaborate the results and the significance of our demonstration.

2.1 TPU Systolic Array

DNNs utilize multiple layers of computations during the inference stage. The multiple layers of DNNs are translated into a matrix multiplication of the activation input, with the weight matrix. A TPU employs a systolic array of $n \times n$ MAC units to expedite the matrix multiplication operations. Figure 6 shows a TPU–the yellow array, where each yellow box is a MAC unit comprising a multiplier and an accumulator. Activation inputs are stored in a unified buffer and subsequently streamed across the MAC array, while the weight matrices are pre-loaded into the MAC units. The activation inputs and stationary weights maintain an 8-bit precision.

2.2 Impact of PV at Low-Power: LP-Faults

PV sensitivities experienced at STC are severely exacerbated at LPC. Therefore, a small effect of PV in a post fabrication circuit can produce a substantially large delay variation, when the operating region is switched from STC to LPC. Although the actual variation in physical dimensions (i.e., $gate$ length or t_{ox} length) remains identical, the observed variation will be in the transformed or elongated delays. Hence, any sensitized circuit path that exceeds the guard-banded clock period leads to a timing violation. These faults are termed as ***LP-faults*** or ***Delay Faults***. LP-faults can be completely hidden at STC, as the timing guard-band essentially covers up the relatively smaller delay variations (Figure 1). As we move toward LPC, the voltage and frequency are lowered appropriately, and a suitable timing guard-band is applied. However, as the delay variations are significantly higher at LPC, the prolonged combinational delay due to PV will exceed the timing guard-band and trigger an LP-fault. Furthermore, as factors leading to LP-faults emanate during fabrication, their effects can be perceived only during the real-time working environment.



(a) Zero Weight—Masks LP-fault. (b) Non-zero Weight—Exposes LP-fault.

Fig. 2. Sensitized paths in a MAC unit for zero and non-zero weights, respectively.

2.2.1 *Threats Due to LP-Faults.*

Effect on Inference Accuracy: In a TPU systolic array (at every clock cycle), the output of each MAC unit, is added to the resulting activation and weight product of the consecutive downstream MAC unit. Hence, the accumulated product from the lowermost row forms a single element of the output matrix. However, as the systolic array is operated at LPC, the effects of PV in a MAC unit can instigate a delay fault and produce an erroneous output. As the PV-affected gates are distributed across the systolic array, more and more MAC units will be rendered faulty. Consequently, a large magnitude of LP-faults will increase the propagation of erroneous outputs and eventually incorrect values will be stored in the output matrix (later shown in Figure 9). Therefore, inference accuracy processed using the erroneous values will be significantly lowered.

Significance of Zero Activation: Figures 2(a) and 2(b) compare the sensitized paths in a MAC unit between a zero weight and a non-zero weight. A zero weight in the MAC unit drives the output of the multiplier to zero for the entirety of the matrix multiplication operation, thus sensitizing the second accumulator input path (i.e., output from the upstream MAC) to the output. Eventually, the upstream MAC output is forwarded to the MAC unit in the immediate lower row. Hence, a zero weight masks the LP-faults. However, for a non-zero weight, the circuit path from the activation input will be sensitized. *Consequently, even for a zero activation input, a prolonged delay needed to stabilize the faulty multiplier output can eventually expose an LP-fault, thereby resulting in a non-zero output.* We term such computations as *Fault Prone* zero computations.

We present the motivational data to demonstrate the serious effects of LP-faults, by employing a rigorous cross-layer methodology as described in Section 4.

2.3 Results

Figure 3 depicts a drop in the inference accuracy when the percentage of faulty gates increases in a TPU systolic array, for 7 out of 8 DNN benchmarks. The increase in the LP-fault level challenges the inherent timing error tolerance of DNNs [19], thereby dwindling the DNN inference accuracy. An outlier benchmark is IMDB, where the significant number of zero weights sensitizes the accumulator paths to the output (case of Figure 2(a)), consequently reducing the damaging effects of LP-faults.

Figure 4 shows the percentage of zero computations and *Fault Prone* zero computations for 8 different benchmarks. From Figure 4, it is evident that even though more than 80% of the computations involve zero computations in 7 out of the 8 benchmarks, more than 20% of the zero

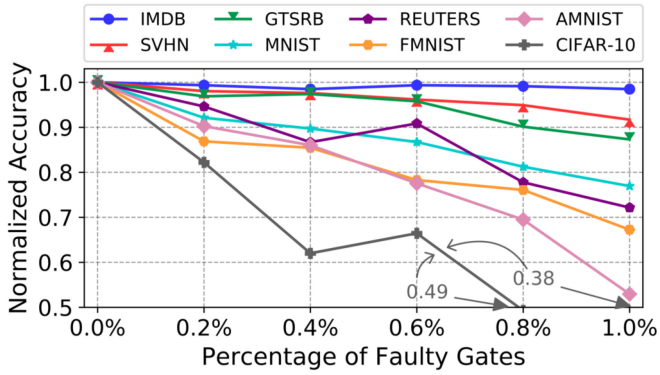


Fig. 3. Increasing number of faulty gates increases the magnitude of LP-faults, thereby deteriorating the inference accuracy.

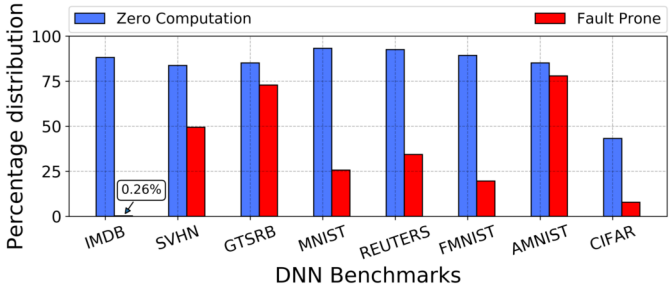


Fig. 4. Zero computations and Fault Prone zero computations elaborated as a percentage of total computations for eight different DNN benchmarks.

computations are *Fault Prone* in 6 DNN benchmarks. Therefore, even a zero computation in a PV-affected multiplier can pose a considerable threat and contribute in lowering the inference accuracy.

Figure 5 presents the combinational delay of a multiplier unit for a *Fault Prone* zero computation. The multiplier is tested for a set of all possible *Fault Prone* zero computations which results in a total of 65,025 combinational delay values. The combinational delays are portrayed as a histogram in Figure 5. The ten bars of the X-axis represent the percentage of MAC computations lying in the respective clock cycle intervals. From Figure 5, it is evident that more than 90% of the *Fault Prone* zero computations occupy more than 40% of the clock cycle.

2.4 Significance

Our findings demonstrate that *the effects of PV at LPC can lead to significant deterioration of the DNN inference accuracy.* Additionally, our study shows that even a predictable zero computation is not safe from the threat of an LP-fault. Since the phenomenon leading to a delay fault is born during fabrication, identical chips can exhibit different variations in the sensitized paths. As AI Edge computing is migrating more toward LPC, the emergence of delay faults can severely hamper the DNN predictions. *Hence, we need a runtime mechanism to identify the faulty MAC units when a systolic array is operated at LPC and an efficient design paradigm to reduce the effect of LP-faults.* Since a zero activation input can be identified due to the redundant bit pattern and the output of

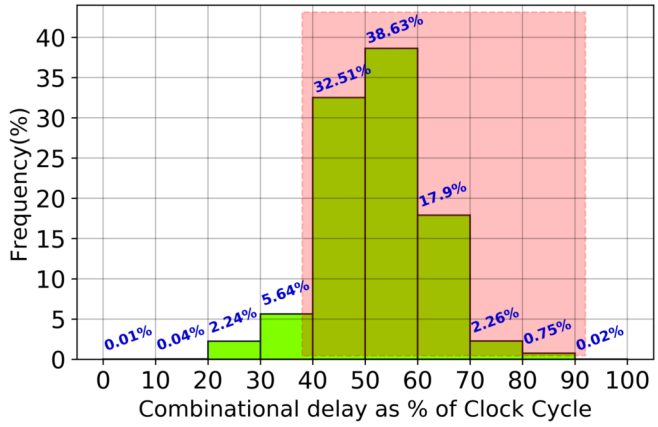


Fig. 5. Combinational delay of a multiplier for Fault Prone zero computations.

a MAC unit for a zero activation is predictable, we can doctor such characteristics to our benefit. With this premise, in the next section, we explore our proposed scheme—STRIVE, to detect and handle LP-faults affected MAC units.

3 Strive Design

In this section, we discuss STRIVE, our novel design paradigm to detect and mitigate the effect of delay faults in an LPC TPU. STRIVE uses a low-overhead *post-fabrication faulty MAC detection technique* to identify the error-prone MACs. Furthermore, by inferring the location of faulty MACs, we devise and employ a *position-aware timing error mitigation* scheme to tackle an LP-fault. We describe the challenges and the overview of STRIVE in Sections 3.1 and 3.2. The threat posed by an LP-fault is explained in Section 3.3. Sections 3.4 through Section 3.6.3 elaborate the design components in detail.

3.1 Challenges

In this section, we highlight the problems which need to be addressed by STRIVE to effectively counter the threat posed by LP-faults.

(1) A faulty MAC unit is susceptible to *Fault Prone* zero computation. Hence, a MAC unit needs to be equipped to handle this scenario (addressed in Section 3.4).
(2) Diagnosing faulty MAC units spread across the TPU systolic array (addressed in Section 3.5.1).
(3) Reclaiming the DNN inference accuracy from an LPC TPU housing PV-affected MAC units (addressed in Sections 3.5.2 and 3.6.3).

3.2 Design Overview

Figure 6 depicts the top-level overview of STRIVE. We enhance the LPC TPU with a **Fault Control Unit** (**FCU**) and each MAC unit with Fault Hop (FHop). Initially, the FCU generates the activation and weight matrices, and the output map. Later, the FCU compares the TPU-generated output matrix and the output map to deduce the location of PV-affected MACs (Section 3.5.1). The faulty MAC locality is later exploited by the FCU, to tackle the LP-faults, using FHop (Section 3.5.2).

Additionally, we discuss an alternate technique Fault Hop Time-Borrow (FHop-TB), to mitigate the effects of an LP-fault, by enhancing the design of FHop (Section 3.6.3).

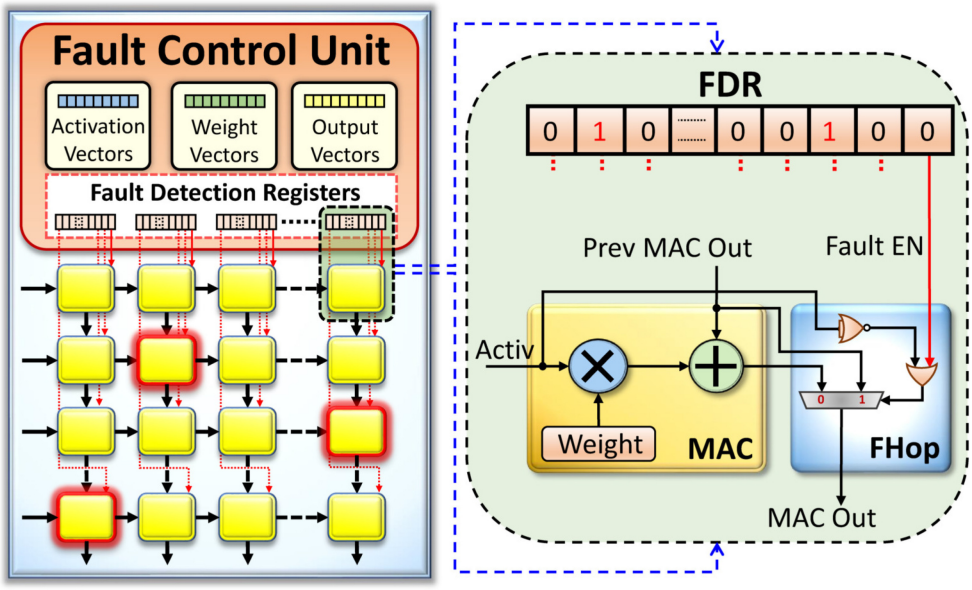
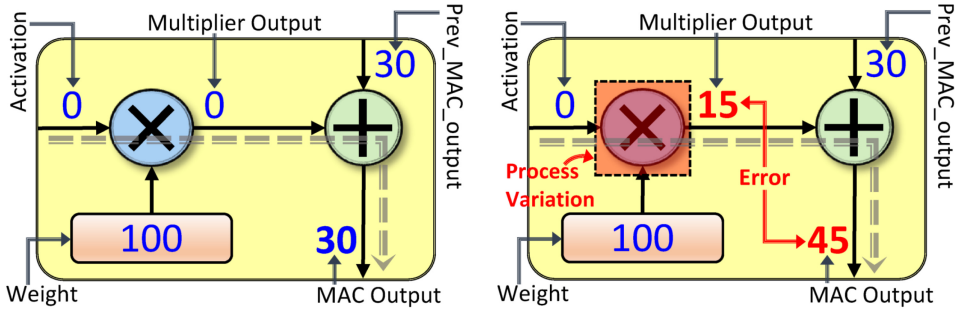


Fig. 6. Design block and dataflow of STRIVE.



(a) Fault free MAC. (b) PV-affected MAC.

Fig. 7. Working of a PV-free and PV-affected MAC.

3.3 Illustrative Example for an LP-Fault

Figures 7(a) and 7(b) describe the operations of a fault-free and faulty MAC, respectively. For a fault-free MAC (Figure 7(a)), the multiplier concludes its computation within the clock period and delivers an error-free output, thereby leading to a correct output from the MAC unit (i.e., ***30***). However, the significant delay stemming from the PV-affected multiplier in a faulty MAC (Figure 7(b)), instigates an LP-fault. Hence, an incorrect value will be processed at the multiplier output (i.e., ***15*** in Figure 7(b)) and an erroneous value (i.e., ***45***) will be delivered to the next logic stage. So, a faulty MAC unit needs to be protected from a zero activation input, which is discussed next.

3.4 Fault Hop (FHop)

In this section, we introduce FHop. As a zero activation input produces a zero computation, we intend to entirely skip the MAC operation for a MAC unit. We augment the MAC unit with a NOR gate, an OR gate and a **multiplexer** (**MUX**), as demonstrated in Figure 6. The MUX output

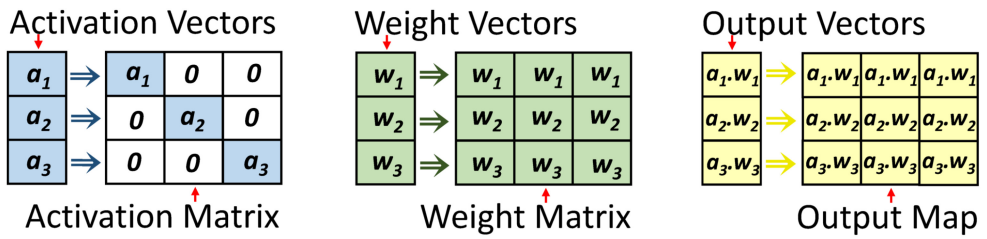


Fig. 8. Development of input matrices and output map—correct outputs—by FCU.

is controlled either by the NOR gate (viz., the zero activation input) or the *Fault EN* signal from FCU (discussed in Section 3.5). Thus, for a *zero activation input or when a Fault EN signal is set*, the erroneous MAC operation is bypassed and the accumulator input is directly presented to the MAC output through the MUX. Hence, *FHop prevents a possible Fault Prone zero computation and aids in the identification of faulty MACs* (discussed in Section 3.5.1.2).

3.5 Fault Control Unit (FCU)

FCU houses the **Fault Detection Registers** (**FDRs**) along with the fault detection vectors. We dedicate one FDR to each column of the systolic array (Figure 6). Each bit of the FDR is labeled as a *Fault EN* signal and maps to a corresponding MAC unit in that column. The FCU operates in two modes: (a) *Fault Detection Mode*, to determine the faulty MACs, and (b) *Fault Resilient Mode*, which is the nominal operating mode of the TPU.

In *Fault Detection Mode*, all the bits of the FDR are reset to zero, to skip a MAC operation only for a zero activation input. The FCU later sets the FDR bits for all the faulty MACs. Hence, during the *Fault Resilient Mode*, the errant MAC operations are also skipped.

Next, we discuss the two operating modes of the FCU.

3.5.1 Fault Detection Mode. We will demonstrate the detection technique for a 3×3 systolic array for illustration purposes, as the same technique will be scaled up to any dimension of the systolic array.

3.5.1.1 Matrix and Map—correct Outputs—generation. Figure 8 depicts the low-overhead matrix/map generation methodology. The FCU generates the activation matrix by allocating the individual activation vectors along the diagonal and zero value for the non-diagonal entries, thereby creating a diagonal matrix. For the weight and output matrix, all the elements in a row are assigned the same weight and output vector. However, each row is allocated a different vector.

3.5.1.2 Faulty MAC detection.

Ideal Systolic Array Operation: Figure 9 demonstrates the cyclewise accurate matrix multiplication between activation and weight matrix for the choke point detection. The systolic array is pre-loaded with the weight matrix, while the activation matrix is transposed and streamed to the individual rows (i.e., from left to right). Activation table and the Output Matrix table shown below the systolic array in Figure 9, coherently maps the activation input being streamed to corresponding MAC units and the output accumulated from the last row of the systolic array in the respective clock cycles.

— In Cycle 1, activation a_1 is multiplied with weight w_1 in the only active MAC.
— In Cycle 2, zero activation values are streamed to the first and second row. The activation a_1, from Cycle 1, is forwarded to the successive column (i.e., second column) of the first row. MAC unit of the first row and first column incurs a zero output, as the zero activation (i.e.,

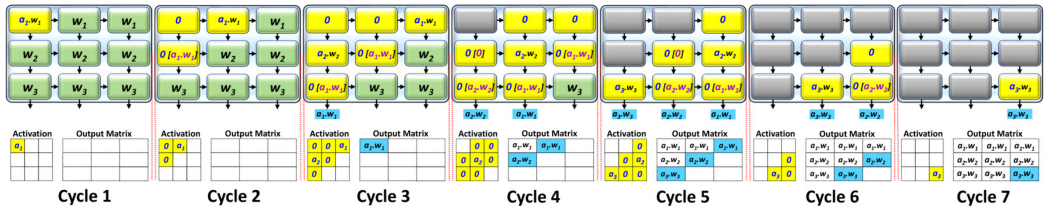


Fig. 9. Data flow pattern in a 3 × 3 systolic array for an ideal scenario. Green MACs are yet to receive the activation, yellow MACs are currently in operation and grey MACs have completed their execution. Only the final operation from each MAC is shown on the yellow MACs for space constraints. "$0[a_n.w_n]$" operation on a yellow MAC indicates that the activation input is "0" and accumulator input "$a_n.w_n$" will be forwarded to the next stage.

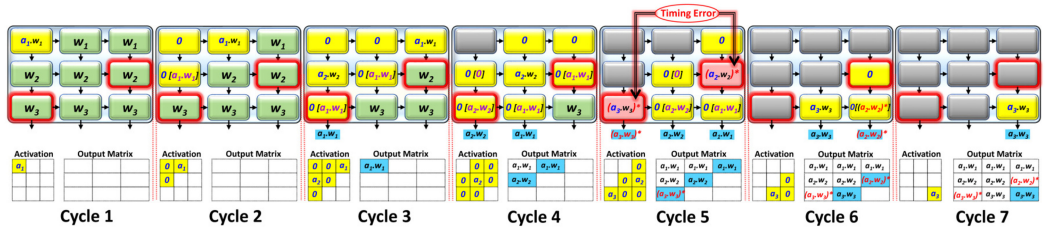


Fig. 10. Faulty MACs are highlighted with a dark red outline. Red MACs denote the occurrence of a timing error.

0) is multiplied with w_1. However, as the MAC units are enhanced with FHop, MAC unit of the second row and first column encountering the zero activation input will skip the MAC operation and forward the upstream MAC output (i.e., $a_1.w_1$) to the downstream MAC unit.
— In Cycle 3, active MAC unit in the last row receives the zero activation and the resulting output (i.e., $a_1.w_1$) is recorded in the appropriate entry of the output matrix.
— From Cycle 4 to Cycle 7, MAC outputs from the last row are collected and synchronously allocated to the respective entries of the output matrix.

Systolic Array Operation with Faulty MACs: Figure 10 illustrates a systolic array with the faulty MAC units.

— Ideal scenario is replayed from Cycles 1 to 4, as non-zero activation has not reached the faulty MACs.
— However, in Cycle 5, activation inputs a_2 and a_3 reach the faulty MAC units in rows two and three, respectively. As the non-zero activation inputs are multiplied by the respective weights, the extended combinational delay will trigger a timing error and erroneous values (i.e., $(a_2.w_2)^*$ and $(a_3.w_3)^*$) are generated. The erroneous value from the faulty MAC unit in the third row (i.e., $(a_3.w_3)^*$) is stored in the output matrix.
— In Cycles 6 and 7, the erroneous value $(a_2.w_2)^*$ along with other output entries are accordingly forwarded and stored in the output matrix.

Detection of Faulty MACs: Figure 11 presents the process of detecting the Faulty MACs, using the output matrix. FCU compares the respective entries of the output matrix with the output map (Figure 8) and determines the location of faulty MACs. FCU thus sets the bits of the FDR targeting the faulty MACs in each column (Figure 6). An entire systolic array operation is utilized by STRIVE for the errant MACs detection process.

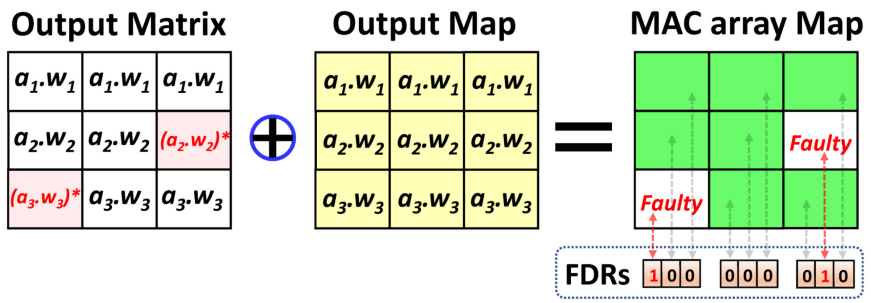


Fig. 11. Comparing the entries of the Output Matrix and the Output Map yields the locality of the Faulty MACs.

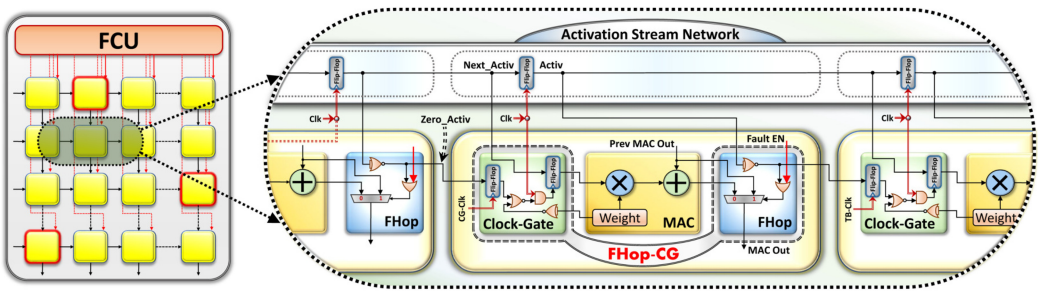


Fig. 12. FHop-CG utilizes clock gating in a MAC unit for a zero activation or zero weight input.

Additionally, the entire fault MAC detection process can be repeated periodically to detect newer faulty MACs emanating from aging as well as due to soft errors.

3.5.2 Fault Resilient Mode. In the Fault Resilient Mode, as the faulty MACs are identified, the set *Fault EN* signal effectively skips the faulty MAC computation. Even though the skipping operation generates a loss in precision for the output matrix entries, the overall inference accuracy is not relatively affected due to the algorithmic level error tolerance of DNNs [19]. However, increase in the number of faulty MACs will eventually lead to a drop in DNN inference accuracy. To address this caveat, we enhance the design of FHop with a *time-borrow* feature to achieve a superior performance, as discussed in Section 3.6.3.

3.6 FHop Variants

We explore the different variants of FHop, to emphasize on energy-efficiency and the improved performance of STRIVE.

3.6.1 Fault Hop Activation (FHop). FHop encompasses the complete architecture discussed so far. The accumulator input is bypassed for zero activation input or the *Fault En* signal (Section 3.4).

3.6.2 Fault Hop Clock-Gate (FHop-CG). This is a variant of FHop, with an augmented clock gating unit to reduce the dynamic power whenever a MAC unit has a zero activation or zero weight input. Figure 12 depicts a MAC unit with FHop enhanced by a **Clock-Gate** (**CG**). As the activation input is streamed from the leftmost MAC unit toward the rightmost MAC unit, *Zero_Activ* signal is generated and delivered to the next MAC unit (i.e., the MAC unit to the immediate right) in each cycle. *Zero_Activ* in each MAC unit is latched using a Clock-Gate clock (CG-clock). CG-clock is 50% delayed from the system clock to secure a stable *Zero_Activ* signal and prevent the transitioning

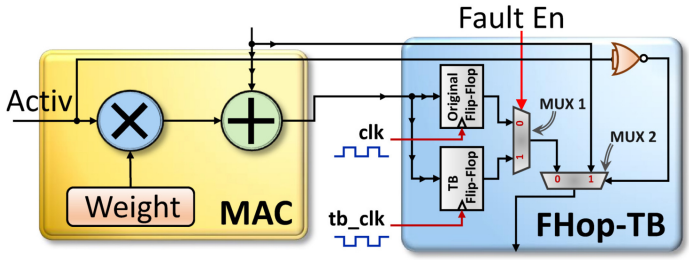
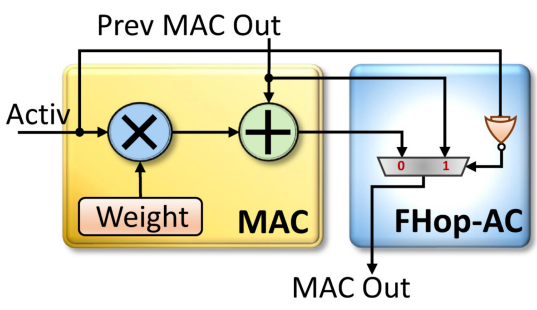
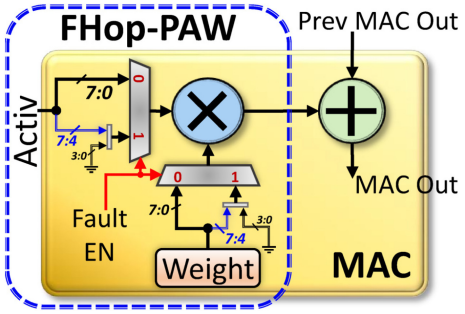


Fig. 13. Detection and correction of timing errors by FHop-TB, using the Time-Borrow technique.



(a) Accumulator input will be forwarded to the next logic stage for a zero activation input by FHop-AC.



(b) Activation and weight inputs are pruned reducing the multiplier delay for a fault prone MAC unit.

Fig. 14. Additional variants of FHop.

of the zero activation input to the MAC unit. Furthermore, for a zero weight input the clock to the MAC unit is gated for the entire systolic array operation.

3.6.3 Fault Hop Time-Borrow (FHop-TB). Figure 13 presents FHop-TB—an enhanced variant of FHop that utilizes the combinational delay disparity between the multiplier unit and the accumulate unit to prevent the propagation of corrupted values. FHop-TB uses a **Time-Borrow** (**TB**) flop to capture the delayed output of the faulty MAC and direct it to the next logic stage within the same clock cycle.

The accumulation process utilizes less than 50% of the clock cycle. For a faulty MAC, we intend to borrow 50% of the clock cycle from its downstream MAC (i.e., the downstream MAC will be performing its own multiplier operation during this period) to procure the correct MAC output and ensure that the correct output is provided to the downstream MAC for its accumulation process. Thus, the TB latch is driven by a delayed clock (i.e., 50% shift from the system clock) to perform the *Time Borrowing* operation. In a faulty MAC, the output of the shadow latch is sensitized to the MUX 1 output; else, the output of the original latch is delivered to the MUX 1 output. For a zero activation, MUX 2 bypasses the upstream MAC output to the downstream MAC unit, irrespective of a faulty/non-faulty MAC unit. Therefore, the output of MUX 2 can switch between MUX 1 output and upstream MAC output.

3.6.4 Fault Hop Activation (FHop-AC). FHop-AC is variant of FHop without the faulty MAC detection/correction scheme. Figure 14(a) depicts the architecture of FHop-AC. FHop-AC ensures the bypassing of the upstream MAC output to the downstream MAC for a zero activation input. STRIVE-AC helps us to better understand the efficacy of a reactive approach in retaining the DNN inference accuracy for a TPU severely affected by LP-faults.

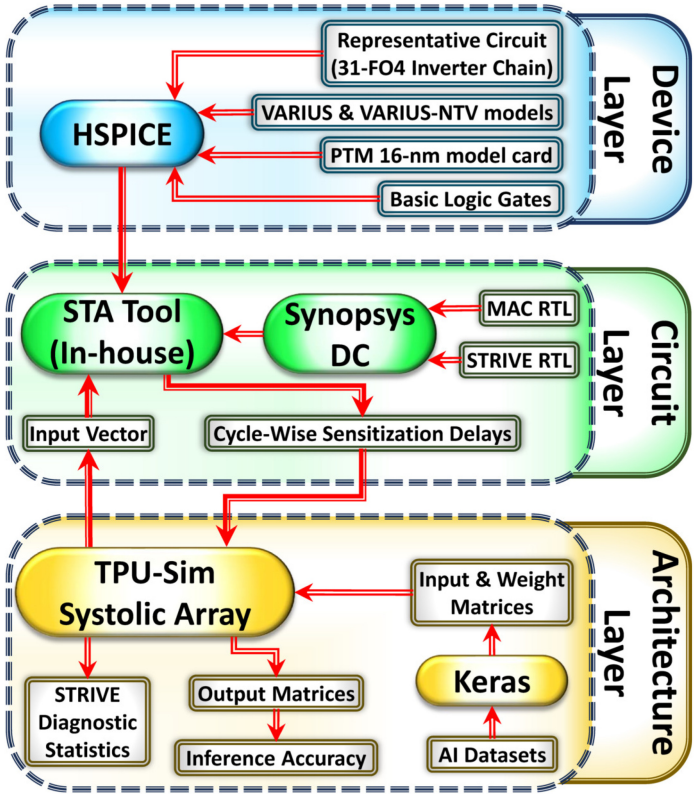


Fig. 15. Cross layer methodology.

3.6.5 Fault Hop Pruned Activation-Weight (FHop-PAW). FHop-PAW is a pruned variant of FHop which prunes the activation and weight inputs for the faulty MAC units. Figure 14(b) presents the overview of FHop-PAW. Whenever the Fault EN signal is set for the faulty MAC unit, the pruned inputs are directed toward the multiplier. The lower 4 bits out of the 8-bit activation and weight inputs each are set to zero. Hence, only the upper 4-bits of the multiplier inputs are connected. As the lower half bits are constant for the entire systolic array operation, lesser paths are sensitized in the multiplier which reduces the combination delay of both the error-free and error-prone MAC units by 20%. However, the increase in the speed of the computation is accompanied by a drop in precision of the output, which eventually affects the inference accuracy.

4 Methodology

Our extensive cross-layer methodology allows us to combine functional simulation of a DNN inference task (thus, allowing precise estimation of inference accuracy) with a holitic power-timing characteristics spanning three layers: device, circuit, and architecture. Figure 15 depicts the cross-layer methodology employed by STRIVE.

4.1 Device Layer

We perform HSPICE simulations on basic logic gates (e.g., NOR, NAND, and Inverter) using the 16-nm predictive technology model, to measure their delay distributions [54]. We use VARIUS-NTV to implement the impacts of with-in die PV at LPC [21]. We incorporate the FinFET

Table 1. List of DNN Benchmarks and Their Error-Free Accuracy

Benchmarks		Error-free Accuracy
Name	**Architecture**	
IMDB [33]	CONV: 400x(400x50)x(398, 256), FC: 256x1	0.89
SVHN [35]	CONV: (32, 32, 3) × (32, 32, 32) × (32, 32, 32) × (14, 14, 64) × (14, 14, 64) × (5, 5, 128) × (5, 5, 128), FC: 512 × 512 × 10	0.94
GTSRB [45]	CONV: (3, 48, 48) × (32, 48, 48) × (32, 46, 46) × (64, 23, 23) × (64, 21, 21) × (128, 10, 10) × (128, 8, 8), FC: 2048 × 512 × 43	0.97
MNIST [27]	FC: 784 × 256 × 256 × 10	0.98
REUTERS [2]	FC: 2048 × 256 × 256 × 46	0.80
FMNIST [50]	FC: 784 × 256 × 512 × 10	0.89
AMNIST [1]	CONV: (20, 25, 1) × (20,25,128) × (20,25,64), FC: 32000 × 256 × 128 × 40	0.92
CIFAR-10 [24]	CONV: (32, 32, 3) × (32, 32, 32) × (32, 32, 32) × (16, 16, 64) × (16, 16, 64) × (8, 8, 128) × (8, 8, 128), FC: 2048 × 512 × 10	0.77

attributes using VARIUS-TC model [22]. The propagation delays are then used in the circuit layer (Section 4.2) to realize the combinational delays due to LP-faults, as well as, the nominal delays.

4.2 Circuit Layer

We implement the TPU systolic array, augmented with the components from our design in Verilog RTL. The developed RTLs are synthesized using Synopsys Design Compiler. We utilize the synthesized netlists in our in-house statistical timing analysis—STA tool. Using the libraries of the delay distributions for basic gates from HSPICE simulations (Section 4.1), the STA tool generates the sensitized path delays of the MAC unit for all the dataset driven inputs. In addition, the sensitized path delays for the PV affected MAC units are also produced by the STA tool. We use Cadence SoC Encounter to place and route the design and measure the area, power, and wirelength overheads.

4.3 Architecture Layer

We use our in-house cycle-accurate TPU systolic array simulator, modeled on detailed TPU architecture [20]. To accurately simulate a timing violation, the combinational delays for a PV affected MAC unit and a nominal MAC unit, developed from the STA tool is integrated into the TPU simulator. TPU simulator is interfaced with Keras to recreate a real-time DNN inference platform [9]. Initially, we train the DNN benchmarks using Keras (running tensorflow in the background). Table 1 lists the DNN benchmarks along with their error-free accuracies. We extract the activation input from each layer and weight inputs from the trained model weights. The extracted inputs are separated into 256 × 256 matrices. The corresponding pair of activation and weight matrices are provided as inputs into the TPU simulator. The output matrices are appropriately combined together to obtain the inference accuracy.

5 Experimental Results

In this section, we examine the efficacy of STRIVE at LPC operating conditions—a typical use case for an edge system deployed for an inference task. The baseline operation is set to (0.45V, 67.5 MHz) and guarantees an error-free execution for an LPC TPU (i.e., TPU without any faulty MACs). Section 5.1 introduces the comparative schemes. Sections 5.2 and 5.3 elaborate the inference accuracies and energy efficiency of the schemes. Section 5.4 discusses the overheads of STRIVE.

5.1 Comparative Schemes

- **Fault-Aware Pruning (FAP):** This scheme prunes all the weights of faulty MAC units [53]. The weights are implemented for multiple precision values to bypass the faulty multiplier in the MAC unit. FAP does not have an error detection scheme as we have proposed, and therefore *presumes to have an oracular knowledge of the faulty MACs*. As expected for an edge system deployed for inference in the field, we do not expect weights to be retrained, and thus consistently model no retraining for both FAP and our schemes, STRIVE, STRIVE-CG, and STRIVE-TB.
- **STRIVE:** This is our proposed technique, which utilizes the locality of the PV-affected MAC or a zero activation input to skip the erroneous MAC operation. FCU enables the *Fault EN* signal for the respective error-prone MAC to restrict the erroneous data from latching on to the accumulator output. FHop is used to perform the bypassing operation (Figure 6).
- **STRIVE-TB:** This scheme is an upgraded variant of STRIVE, which uses FHop-TB (Figure 13). The error-free output from a faulty MAC unit is captured by the Time-Borrow Flop using the delayed clock, and the *Fault EN* signal enables the forwarding of this correct data to the next stage (Section 3.6.3).
- **STRIVE-CG:** This is a variant of STRIVE with the clock gating scheme, to decrease the dynamic power consumption during the zero weight/activation inputs (Section 3.6.2).
- **STRIVE-AC:** This is a lighter variant of STRIVE with the absence of a faulty MAC detection/correction scheme. The FCU is removed and the MAC operations are bypassed only for a zero activation input (Section 3.6.4).
- **STRIVE-PAW:** This is a pruned variant of STRIVE which uses pruned input bits for multiplication. For faulty MAC units only the upper 4-bit MSBs are considered for the multiplier inputs (i.e., both activation and weight inputs) (Section 3.6.5).

5.2 Inference Accuracy

Figure 16 depicts the normalized inference accuracy for the five comparative schemes, as the percentage of faulty gates are increased in the TPU. STRIVE-CG is not included in Figure 16 as it an enhancement to STRIVE only on the aspect of power savings and uses the same inference engine. All the schemes are operated at the baseline voltage and frequency. The X-axis represents the percentage of faulty gates in the TPU. STRIVE and FAP are able to offer modest error resilience for 4 out of first 6 DNN benchmarks up to 0.6% fault rate. In case of AMNIST and CIFAR, FAP and STRIVE are able to retain the inference accuracy for a fault rate up to 0.2%. For REUTERS, FMNIST, AMNIST and CIFAR the extreme data-delay variance between the activation sequences causes significant timing errors, thereby dropping the inference accuracy for STRIVE and FAP. Both STRIVE and STRIVE-CG exhibit identical performance, as both are equipped with FHop and FCU to tackle timing errors. The significant number of zero activation computations along a with an effective faulty MAC detection/correction mechanism aids STRIVE in retaining the inference accuracy as the percentage of faulty gates are increased. STRIVE-AC has a steady decline in inference accuracy due to the absence of a FCU as the faulty MACs are only protected during zero activation sequences. These results indicate the importance of possessing a faulty MAC identification mechanism in reclaiming the DNN inference accuracy. STRVE-PAW operates on par with FAP and STRIVE for the initial 5 DNN benchmarks. As the data-delay variance is higher in the remaining benchmarks, loss of precision due to pruning takes precedence and consequently the inference accuracy deteriorates. However, STRIVE-TB incurs less than 1% loss in inference accuracy for up to 1% gate fault rate in a TPU, as it is able to capture the delayed data using the *Time-Borrow* approach. The error resilience

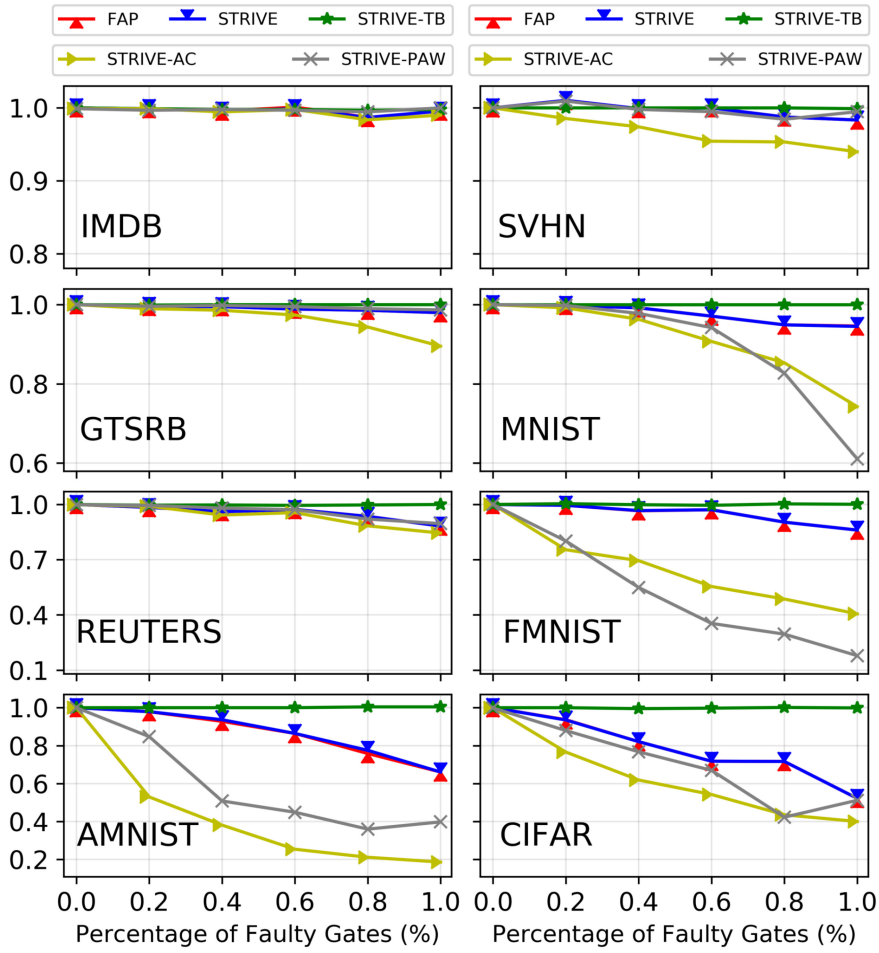


Fig. 16. Normalized Inference accuracies of FAP, STRIVE, STRIVE-TB, STRIVE-AC, and STRIVE-PAW across 8 DNN benchmarks for different percentages of faulty gates (**under low-power**) affected in a TPU systolic array. The Y-axis values are normalized to the corresponding error-free accuracy at the baseline LPC TPU operation.

of STRIVE schemes from highest-to-lowest is STRIVE-TB →STRIVE→STRIVE-PAW→STRIVE-AC. *Overall, STRIVE-TB vastly outperforms FAP, demonstrating remarkable resilience in retaining inference accuracy under LP-faults.*

5.3 Energy Efficiency

Figure 17 presents the energy efficiencies of the comparative schemes measured using the **Tera Operations Per Second (TOPS)**/Watt metric. TOPS measure will be the same for all the schemes at the corresponding frequency of operation. The TOPS/Watt is normalized to that of the LPC TPU, operated at the baseline operating conditions. FAP has a lower energy efficiency due to its larger power consumption in comparison to STRIVE variants. FAP uses multiple MUXs and a comparator in each MAC unit for bypassing and pruning purposes which increases its power consumption [53]. STRIVE boasts a higher energy efficiency due to its low overhead operation compared with

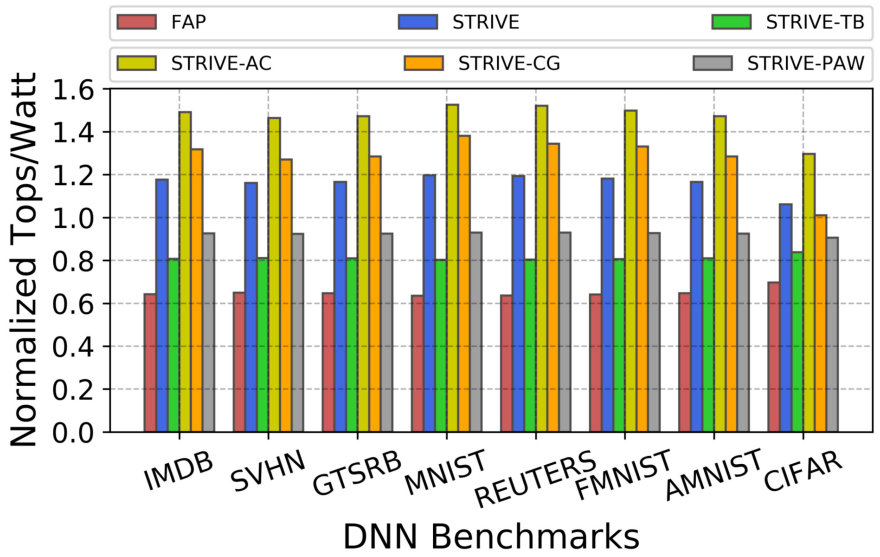


Fig. 17. Energy Efficiency comparison of FAP, STRIVE, STRIVE-TB, STRIVE-AC, STRIVE-CG, and STRIVE-PAW (higher is better).

STRIVE-TB. Due to the exclusion of the error detection mechanism and a minor architectural overhead, STRIVE-AC achieves better energy efficiency. However, STRIVE-CG enhanced with the clock gating scheme outperforms the other STRIVE variants which enhance performance. STRIVE-PAW offers slightly better energy efficiency due to the reduced circuitry (i.e., shadow flop and NOR gate) in comparison to STRIVE-TB. STRIVE, STRIVE-TB, STRIVE-AC, STRIVE-CG and STRIVE-PAW offers an average 1.8×, 1.2×, 2.3×, 2.0×, and 1.4× better performance per unit power, compared with FAP. The energy efficiency of STRIVE schemes from highest-to-lowest is STRIVE-AC→STRIVE-CG→STRIVE→STRIVE-PAW→ST RIVE-TB. STRIVE-TB with a clock gate mechanism offers an average 1.7× increased energy efficiency. Hence, *STRIVE is an energy efficient design paradigm, able to extract superior performance even from an error-prone TPU.*

5.4 Implementation Overheads

STRIVE incurs overheads due to the inclusion of FCU and the combinational logic for FHop or FHop-TB. Area overhead added by STRIVE and STRIVE-TB is ∼ 1.8% and ∼ 6%. STRIVE and STRIVE-TB incurs a power overhead of ∼ 2% and ∼ 5%, and a wire-length overhead of ∼ 1.1% and ∼ 3.5%, respectively. Area and wire-length overheads for STRIVE-CG due to FHop and Clock-Gate are ∼ 4.4% and ∼ 1.4%, respectively. STRIVE-CG consumes ∼ 1% less power due to the clock gating scheme. Area, power, and wirelength overheads incurred for STRIVE-AC are ∼ 0.2%, ∼ 0.4% and 0.5%, respectively. STRIVE-PAW has an area, power and wirelength overhead of ∼ 2.2%,∼ 3.1%, and ∼ 1.3%.

5.5 Test Pattern Generation and Utilization

The input bit-width for activation and weight is 8-bit each. Hence, for a MAC unit with a stationary weight and a transitioning activation input at every cycle, there can be a total of 16,777,216 combination delay values (i.e., 256[*weights*] × 256 [*prev_activ*] × 256 [*curr_activ*]). For the test input generation, we need to only consider the delays when activation vector transitions from a *zero to non-zero* value for all the *non-zero* weights (Figure 10). Hence, we only need 65025 (255 × 255)

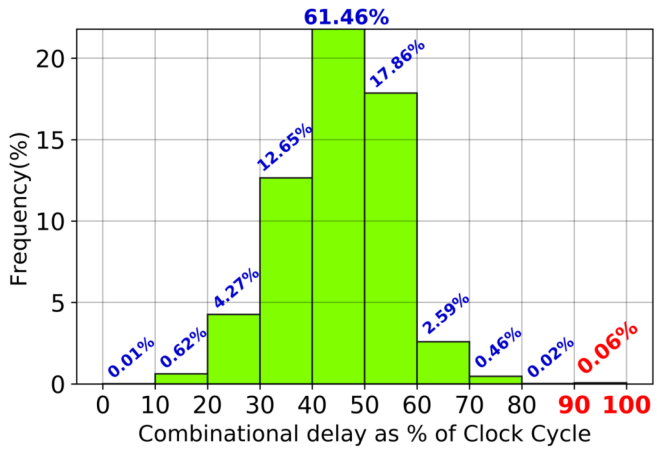


Fig. 18. Combinational delay for non-zero weight and *zero to non-zero* activation transition.

Table 2. Diversification of STRIVE Usage

APPLICATION	SCHEME
Sparse Datasets	STRIVE, STRIVE-AC, STRIVE-PAW
High Performance	STRIVE-TB
Low Power Consumption	STRIVE-CG, STRIVE-TB(CG)

delay samples. Figure 18 presents the delay profile for these 65025 samples. Next, we only consider the activation and weight pairs which consume more than 90% of the clock period. Therefore, we require only *0.06% of the entire 65025 pairs* (i.e., indicated in red color in Figure 18), which accounts up to only ***42 activation-weight pairs***. Based on our investigation using our in-house STA tool, we could confirm that these patterns cover most of the paths in the MAC circuit. Hence, any paths encountering extreme PV will be activated by these minimal test patterns and result in a timing violation thereby, aiding the fault detection process. To detect the location of a fault prone MAC unit we need an entire systolic array operation. For our approach, since each MAC unit is tailored to encounter only one non-zero product, we can reuse the same activation-weight pairs from the existing subset for MAC units on multiple rows. Additionally, we randomize the pairs for each iteration. Due to the intense time involved in the cyclewise simulations, we restricted ourselves to 5 iterations. As most of the paths are sensitized by the high delay vectors we are able to obtain 100% coverage in faulty MAC localization within 3 iterations.

5.6 Scheme Usage

In this article, we have introduced four variants of STRIVE. Additionally, we have implemented a clock gated version of STRIVE named STRIVE-CG. We can also use a clock gated version of STRIVE-TB by including the same clock gate as depicted in Figure 12. By analyzing the power and performance results we can utilize different variants of STRIVE for specific applications. Table 2 presents the relevant STRIVE schemes which can be used for different applications. For applications having significant amount of zero activations, skipping the MAC operations will have minimal effects on the inference accuracy. Hence, STRIVE-TB can be avoided in this application. For medical applications and datacenters where performance cannot be compromised, STRIVE-TB can be employed. Clock-Gate version of STRIVE can be utilized whenever power is of prime concern.

STRIVE addresses the fault-detection and fault-handling in two separate stages. Initially, the faulty MAC locality is procured (Section 3.5.1). As the LP-faults will emanate based on the operating conditions, identifying the faulty MACs before the application processing becomes vital. Hence, the fault detection process is independent of the application which will be processed on the hardware and all the STRIVE schemes (except STRIVE-AC) will use the same fault detection mechanism. However, as the STRIVE schemes offer various degrees of protection based on the application under consideration and the processing efficacy requirement, STRIVE schemes can be deployed based on the usage (Table 2).

5.7 Impact of STRIVE on Different Faults

In the semiconductor testing process, hard faults (i.e., permanent faults like stuck-at-faults) are normally detected immediately after manufacturing using scan chains and ATPG. In case of the systolic arrays, using the conventional testing methodologies we can determine the functional correctness after fabrication. However, the soft errors (i.e., transient faults or LP-faults) only manifest at unexpected circumstances or lower voltages and more specifically after chip deployment. Enhancing the systolic arrays with STRIVE will enable the system to detect hard faults (which can occur due to long time usage) as well as soft errors on deployed machines using smaller run times. Additionally, LP-faults can manifest in any random MAC unit in different chips. *As DNN weights can be appropriately trained by using layer-mapping to individual MAC units, faulty MAC localization information provided by STRIVE can aid in training the applications specifically used in a particular chip.* Hence, by leveraging the fault PE localization information we can extract maximum performance even from the faulty systolic array affected by different kinds of faults.

6 Related Works

A vast amount of research has been conducted in the AI computing realm. Section 6.1 highlights the extensive research works based on the multiple aspects of the architecture. Section 6.2 compares the schemes whose error detection capabilities are closer to our technique.

6.1 Relevant Research

In this section, we discuss various works in AI domain based on the different aspects of the design from Sections 6.1.1 to 6.1.5. Additionally, Table 3 provides an extensive list of all the works in each respective research field.

6.1.1 Fault Tolerance. Several techniques have been explored to mitigate the adverse effects of faulty **Processing Elements (PEs)** in deep learning accelerators using hardware/software level fault tolerance. D.Xu et al. proposed a hybrid computing architecture to recompute the operations mapped to the faulty PEs in arbitrary locations using dot-production processing units [51]. Spyrou et al. demonstrated a fault-tolerant Spiking Neural Network architecture with simplified error detection and recovery scheme [44]. Pandey et al. proposed a timing error resilience technique for a systolic array by boosting the operating voltage of the rows encountering an error causing activation sequence [37].

6.1.2 Underscaling. Researchers have also explored the reduced-voltage operation of multiple components in AI hardware accelerators and their corresponding reliability behavior. Salami et al. evaluated an undervolting technique for Neural Network acceleration in **Field Programmable Gate Arrays (FPGAs)** to improve the power-efficiency [41]. Givaki et al. experimentally evaluated the effect of aggressive voltage underscaling of block RAMs in an FPGA by emulating the real fault maps of SRAM memories [15]. Tang et al. investigated the impact of GPU dynamic voltage and frequency scaling on the energy consumption and performance of DNNs [48]. Elbtity et al.

Table 3. Works Categorized on the Domains

Domain	Works
Fault Tolerance	[12, 37, 44, 51, 52]
Underscaling	[3, 13, 15, 38, 41, 48]
Edge	[8, 11, 29, 30, 32, 34]
Energy Efficiency	[16–18, 23, 36, 40]
Multiple Architectures	[10, 28, 31, 43, 46, 47, 49]

introduced approximate PEs to replace the direct quantization of inputs and weights, utilized per-approximate units and shared them with approximate PEs to reduce the overhead arising from the element-wise operation [13]. Reagen et al. developed a fault injection framework to perform fault analysis on DNNs and validated an orders of variation in the fault tolerance based on structure, model, and layers for DNNs [38].

6.1.3 Edge AI. A recent trend have been observed leading to extend DNN applications to platforms that are more resource and energy-constrained, e.g., edge/mobile. As DNN requires a large amount of computation and memory access, Edge DNN acceleration have been very challenging. Chen et al. aimed to reduce the size of DNNs resulting in more compact structures and/or have high data sparsity to improve the hardware processing efficiency [8]. Lee et al. explored optimization methods for hardware architectures for energy-efficient DNN processing on edge devices [29]. Marty et al. utilized the algorithmic properties of CNNs to propose a light weight error detection technique to enhance the efficiency of a hardware accelerator [34].

6.1.4 Energy Efficiency. Different techniques have also been explored to retain the energy efficiency in the DNN hardware. Nguyen et al. presented a stretchable DRAM refresh control technique to replace non-critical bits with parity bits of error correction schemes resulting in improved energy efficiency without performance degradation of DNNs [36]. Koppula et al. proposed a general framework that reduces DNN energy consumption by using approximate DRAM devices while meeting the user-specified DNN accuracy requirements [23]. Jiang et al. developed a **Dynamic Voltage and Frequency Scaling (DVFS)** framework on FPGAs to analyze the impact of DVFS on **Convolutional Neural Networks (CNNs)** in terms of performance, power, energy efficiency, and accuracy [18]. Ruospo et al. reduced the fault simulation times for DNN inference execution using a fault injector framework, which replicates the pipeline flow [40]. Hosseini et al. proposed algorithms to approximate the value of fault-free bits in the defective DNN weight memories and reduce the drop in DNN accuracy due to faulty Non-volatile memories [17]. Hoang et al. demonstrated a technique to develop a unified fault map from smaller fault maps lower the retraining rounds and reduce the retraining overhead [16].

6.1.5 Error-Handling in Different Architectures. Wider research has been conducted to perform error detection in different architectures. Sullivan et al. utilized the register file ECC hardware and without the eliminating the capability of ECC to identify and rectify storage errors detected pipeline errors in GPUs [46]. TU et al. proposed a methodology to combine address symbolization, symbolic-concrete memory map and symbolic memory tracking to efficiently detect temporal memory errors [49]. Kim et al. presented an error detection technique for GPGPUs by re-executing the instructions to evaluate the prediction accuracy of the possible errors and rectifies the erroneous predictions [31].

While these techniques emphasize on improving the timing error resilience of hardware accelerators using algorithmic and timing speculation methodologies, our proposed technique aims to

Table 4. Comparison of Threats with Other Schemes

Works	Permanent Faults	LP/Transient Faults	Usage after deployment	Faulty PE Localization	
				Pre-Deployment	Post-Deployment
C-Testing and Fault Localization [6]	✔(H/W)	✖	✖	✔	✖
Black Box Bayesian Optimization [5]	✔(S/W)	✖	✔	✖	✖
FUSA Violation Detection [25]	✔(S/W)	✖	✔	✖	✖
GAN Point [26]	✔(S/W)	✖	✔	✖	✖
STRIVE	✔(H/W)	✔	✔	✔	✔

identify the erroneous MAC units in the TPU using the systolic dataflow and tailored inputs, and mitigate them using architectural augmentation.

To the best of our knowledge, *this paper is the first work that addresses the threat of a delay fault in altering the output of a zero multiplier computation to a non-zero value, and introduce a novel post-fabrication dynamic timing error detection scheme for a TPU affected by faulty MAC units due to LP-faults.*

6.2 Comparison with Other Fault Detection Schemes

Table 4 provides a broad review of the error detection capabilities of the comparative schemes with STRIVE. We have thoroughly explained the details of these works and the differences with our technique as follows.

(1) Chaudhuri et al. introduced a fault localization mechanism in AI accelerators using a checkerboard style method to generate and propagate test patterns, and proposed a priority encoder based localization method to identify the fautly PEs [6]. The technique operates on a subarray of PEs, excluding the PEs on the top and bottom rows, in addition to the PEs on the leftmost column. Test patterns for PEs in the subarray are generated based on the logic in the adjacent PEs. The external control flops are allotted to a scan chain and shared by the PEs. The dataflow is controlled in multiple iterations and the respective output is obtained. For fault localization, the bypass signal generator with appropriate inputs is used to issue the bypass signals and a priority encoder is used to output the prioritized faulty PE cluster ID.
Comparison: This work uses scan chains to develop the test patterns and the dataflow is directed by control signals to eventually detect the faulty PEs. Next, the PEs on the top/bottom row and leftmost column are not covered by the checkerboard style technique, but using the top-off patterns. Hence, additional logic circuitry is required for different batches of PEs. However, STRIVE uses the normal systolic dataflow without any additional control signals to localize the faulty PEs. Additionally, as scan chains become unavailable after the chip deployment, any LP-faults which only appear when the voltage is down-scaled will be completely hidden. Furthermore, faults can appear on different PEs in different chips. As STRIVE employs a dynamic approach, faulty MACs can be localized even after deployment by running the TPU in the fault detection mode. Hence, a designer possessing the fault locality (which varies for different chips) information can train the benchmarks accordingly for different deployed chips thereby, improving the overall inferential capability of the accelerator.

(2) Chaudhuri et al. achieved black-box optimization using a **Bayesian Optimization** (**BO**) algorithm as an engine and proposed a test generation platform for systolic array accelerators [5]. The black-box optimization engine designed using the BO algorithm, develops test images for the systolic array with an objective to distinguish between an erroneous and error-free systolic array. The classification accuracy of the faulty and fault-free systolic array is used to eventually separate them. The BO-sampled images are encoded and decoded between a low-dimensional latent space using a Non-negative matrix factorization technique.

The image is sampled in the latent space in each iteration, before being decoded toward the prescribed dimensional space and then the fault simulation is performed on the accelerator. *Comparison: In this scheme, the test images are generated using an AI training based model. However, our scheme generates test patterns based on the detailed analysis of the arithmetic units (Section 5.5). Additionally, STRIVE can be employed for both hard and soft faults. Furthermore, our scheme can pip-point the faulty PEs which can aid in retraining purposes as the layerwise operation of DNNs can appropriately scale the weights of the MAC units adjacent to the faulty MACs.*

(3) Kundu et al. present two methodologies to detect **Functional Safety** (**FuSa**) violations in a DNN accelerator using minimal test patterns [25]. The authors perform a vulnerability analysis based on faults manifested in bits, units, inputs, DNN layers, neurons, and datasets. Based on these analyses, authors determine that not all faults cause a FuSa based violation in a DNN accelerator and propose two algorithms. The first algorithm is independent of the model under consideration and architecture. The algorithm selects the vulnerable test patterns which are most probable to be classified incorrectly due to the presence of faults. Furthermore, the similarity quotient for the test patterns in determined based on the Euclidean distance between the datapoints under consideration. The second algorithm is model-aware technique and selects the test patterns which are predicted correctly, however with a lower confidence score. As the induction of faults into a DNN systolic array affects the confidence value, the prediction confidence becomes vital for the model-aware algorithm.
Comparison: Test patterns in work are generated based on model-aided and model-independent methodologies. However, our scheme segregates the activation and weight input pairs generating the highest combinational delays (Section 5.5). Hence, independent of the benchmarks, the approach employed by our scheme can be dynamically used on any platform and for any application. While the effects of PV can unpredictably increase the combinational delays, the input patterns generating the larger delays will remain the same irrespective of the region of operation. Hence, as the minimal test inputs which are chosen for fault detection inherently produce higher delays (i.e., within the clock period), under the effect of PV will produce timing violations. Additionally, our scheme possesses the faulty MAC localization capability to exclude the erroneous computation from the output accumulation.

(4) Kundu et al. proposed a DNN model independent **Generative Adversarial Networks** (**GANs**) based functional test pattern generation framework to detect FuSa violations in DNN accelerators deployed on field [26]. Permanent faults in MSBs of randomly selected weights are induced using a fault injection framework. A set of test patterns incorrectly classified due to the presence of faults but have been accurately predicted by a faultless accelerator are utilized. The **Deep Convolutional GAN** (**DCGAN**) framework develops a model based on a random noise input and obtains an output having the same dimension as the dataset images, by consistent forward sampling though the layers. Next, the model is trained and deployed, and a set of new patterns absent from the dataset are created by the generator.
Comparison: A cloud trained model for a specific dataset is used in this work to obtain newer test patterns to eventually verify the functional safety of the accelerator. However, STRIVE employs a hardware based approach delving deeply into the circuit-architectural level details to generate test patterns for fault detection. While, both schemes require the normal operation to determine fault characteristics, our scheme avoids the training time needed to generate the test set and is agnostic to the application being run on the accelerator. Additionally, as our scheme provides an information on the faulty MACs, the application to be deployed on the accelerator can be trained using the PE mapping information to obtain better inferential accuracy.

Works [5, 6, 25, 26] emphasize primarily on fault coverage explicitly based on the test vectors and images generated from scan chains and training based models. However, STRIVE delves on fault coverage by generating the test patterns for fault detection by exploring the circuit-architectural details. Works [5, 6, 25] produces a highest fault coverages of 99.8% with 32 test patterns, 86.7% with 9 test patterns and 100% with 10 test patterns, respectively. Work [26] produces a highest fault coverage of 99.8% with a fault rate of 0.4%. While, STRIVE produces 100% fault coverage with only 3 iterations using any of activation-weight pairs (Section 5.5). Additionally, works [5, 25, 26] consume a significant time on training the models to develop the test patterns. As STRIVE has pre-loaded test vectors (Section 5.5) generating higher combinational delays, the fault detection time is significantly reduced.

7 Conclusion

In this article, we highlight the impact of PV in a TPU systolic array under low-power operation. We demonstrate STRIVE—an energy efficient paradigm to identify and nullify the effect of LP-faults, and reclaim the performance from a TPU systolic array affected by faulty MAC units. STRIVE incurs minimum loss in inference accuracy for a TPU infested by a gate level fault rate of 1%. Additionally, STRIVE delivers 1.8×, 1.2×, and 2.0× better TOPS/watt compared with FAP.

Acknowledgements

This work was supported in part by National Science Foundation grant CNS-2106237. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We thank Zinnia Muntaha Mowri and Andrew Chamberlin for their helpful comments and assistance in improving the quality of the manuscript.

References

[1] 2021. Free Spoken Digit Dataset (FSDD). Retrieved from https://github.com/Jakobovski/free-spoken-digit-dataset. (2021).

[2] 2021. Reuters-21578 Dataset. Retrieved from http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html. (2021).

[3] Guanglei An, De Kanishka, Cheng Hao, Rehan Ahmed, Chriswell Hutchens, and Robert L. Rennaker. 2014. An analog front-end circuit with spike detection for implantable neural recording system design. In *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 881–884.

[4] Maurizio Capra, Riccardo Peloso, Guido Masera, Massimo Ruo Roch, and Maurizio Martina. 2019. Edge computing: A survey on the hardware requirements in the internet of things world. *Future Internet* 11, 4 (2019), 100.

[5] Arjun Chaudhuri, Ching-Yuan Chen, Jonti Talukdar, and Krishnendu Chakrabarty. 2023. Functional test generation for AI accelerators using bayesian optimization. In *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 1–6.

[6] Arjun Chaudhuri, Chunsheng Liu, Xiaoxin Fan, and Krishnendu Chakrabarty. 2021. C-testing and efficient fault localization for AI accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 7 (2021), 2348–2361.

[7] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2016), 127–138.

[8] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308.

[9] François Chollet et al. 2015. Keras. Retrieved from https://keras.io. (2015).

[10] Hüsrev Cılasun, Salonik Resch, Zamshed I. Chowdhury, Masoud Zabihi, Yang Lv, Brandon Zink, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu. 2024. On error correction for nonvolatile processing-in-memory. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 678–692.

[11] Kanishka De. 2014. *Design and Implementation of a Low Power T-Gate Cell Library and Comparison with its CMOS Equivalent*. Oklahoma State University.

[12] Kanishka De, Rehan Ahmed, Cheng Hao, and Chris Hutchens. 2017. A low power digitally assisted analog front end for neural interface with optical reception. In *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*. IEEE, 94–100.

[13] Mohammed E. Elbtity, Peyton S. Chandarana, Brendan Reidy, Jason K. Eshraghian, and Ramtin Zand. 2022. Aptpu: Approximate computing based tensor processing unit. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 12 (2022), 5135–5146.

[14] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev R. Rao, Toan Pham, Conrad H. Ziesler, David Blaauw, Todd M. Austin, Krisztián Flautner, and Trevor N. Mudge. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36*. IEEE, 7–18.

[15] Kamyar Givaki, Behzad Salami, Reza Hojabr, SM Reza Tayaranian, Ahmad Khonsari, Dara Rahmati, Saeid Gorgin, Adrian Cristal, and Osman S. Unsal. 2020. On the resilience of deep learning for reduced-voltage FPGAs. In *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 110–117.

[16] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. 2021. TRe-Map: Towards reducing the overheads of fault-aware retraining of deep neural networks by merging fault maps. In *2021 24th Euromicro Conference on Digital System Design (DSD)*. IEEE, 434–441.

[17] Fateme S Hosseini, Fanruo Meng, Chengmo Yang, Wujie Wen, and Rosario Cammarota. 2021. Tolerating defects in low-power neural network accelerators via retraining-free weight approximation. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 5s (2021), 1–21.

[18] Weixiong Jiang, Heng Yu, Jiale Zhang, Jiaxuan Wu, Shaobo Luo, and Yajun Ha. 2020. Optimizing energy efficiency of CNN-based object detection with dynamic voltage and frequency scaling. *Journal of Semiconductors* 41, 2 (2020), 022406.

[19] Xun Jiao, Mulong Luo, Jeng-Hau Lin, and Rajesh K. Gupta. 2017. An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 945–950.

[20] Norman Jouppi, Cliff Young, Nishant Patil, and David Patterson. 2018. Motivation for and evaluation of the first tensor processing unit. *IEEE Micro* 38, 3 (2018), 10–19.

[21] Ulya R. Karpuzcu, Krishna B. Kolluru, Nam Sung Kim, and Josep Torrellas. 2012. VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. IEEE, 1–11.

[22] S. Karen Khatamifard, Michael Resch, Nam Sung Kim, and Ulya R. Karpuzcu. 2016. VARIUS-TC: A modular architecture-level model of parametric variation for thin-channel switches. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 654–661.

[23] Skanda Koppula, A. Giray Orosa, Roknoddin Azizi, Taha Shahroodi, Konstantinos Kanellopoulos, and Onur Mutlu. 2019. EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 166–181.

[24] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report.

[25] Shamik Kundu, Suvadeep Banerjee, Arnab Raha, Suriyaprakash Natarajan, and Kanad Basu. 2021. Toward functional safety of systolic array-based deep learning hardware accelerators. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 3 (2021), 485–498.

[26] Shamik Kundu, Suvadeep Banerjee, Arnab Raha, Fei Su, Suriyaprakash Natarajan, and Kanad Basu. 2023. Trouble-shooting at gan point: Improving functional safety in deep learning accelerators. *IEEE Transactions on Computers* 72, 8 (2023), 2194–2208.

[27] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. Retrieved from http://yann.lecun.com/exdb/mnist/. (2010).

[28] Hayoung Lee, Jihye Kim, Jongho Park, and Sungho Kang. 2023. STRAIT: Self-test and self-recovery for AI accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 9 (2023), 3092–3104.

[29] Jinsu Lee, Sanghoon Kang, Jinmook Lee, Dongjoo Shin, Donghyeon Han, and Hoi-Jun Yoo. 2020. The hardware and algorithm co-design for energy-efficient DNN processor on edge/mobile devices. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 10 (2020), 3458–3470.

[30] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications* 19, 1 (2019), 447–457.

[31] Hyunyul Lim, Tae Hyun Kim, and Sungho Kang. 2020. Prediction-based error correction for gpu reliability with low overhead. *Electronics* 9, 11 (2020), 1849.

[32] Weison Lin, Adewale Adetomi, and Tughrul Arslan. 2021. Low-power ultra-small edge AI accelerators for image recognition with convolution neural networks: Analysis and future directions. *Electronics* 10, 17 (2021), 2048.

[33] Andrew Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. 142–150.

[34] Thibaut Marty, Tomofumi Yuki, and Steven Derrien. 2020. Safe overclocking for CNN accelerators through algorithm-level error detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 4777–4790. DOI: https://doi.org/10.1109/TCAD.2020.2981056

[35] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

[36] Duy-Thanh Nguyen, Nhut-Minh Ho, and Ik-Joon Chang. 2019. St-DRC: Stretchable DRAM refresh controller with no parity-overhead error correction scheme for energy-efficient DNNs. In *Proceedings of the 56th Annual Design Automation Conference*. 1–6.

[37] Pramesh Pandey, Prabal Basu, Koushik Chakraborty, and Sanghamitra Roy. 2019. GreenTPU: Improving timing error resilience of a near-threshold tensor processing unit. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[38] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.

[39] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 267–278.

[40] Annachiara Ruospo, Angelo Balaara, Alberto Bosio, and Ernesto Sanchez. 2020. A pipelined multi-level fault injector for deep neural networks. In *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 1–6.

[41] Behzad Salami, Erhan Baturay Onural, Ismail Emir Yuksel, Fahrettin Koc, Oguz Ergin, Adrián Cristal Kestelman, Osman Unsal, Hamid Sarbazi-Azad, and Onur Mutlu. 2000. An experimental study of reduced-voltage operation in modern FPGAs for neural network acceleration. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 138–149.

[42] Mingoo Seok, Gregory Chen, Scott Hanson, Michael Wieckowski, David Blaauw, and Dennis Sylvester. 2011. CAS-FEST 2010: Mitigating variability in near-threshold computing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 1, 1 (2011), 42–49.

[43] Umair Saeed Solangi, Muhammad Ibtesam, Muhammad Adil Ansari, Jinuk Kim, and Sungju Park. 2021. Test architecture for systolic array of edge-based AI accelerator. In *IEEE Access*, 9 (2021), 96700–96710.

[44] Theofilos Spyrou, Sarah A. El-Sayed, Engin Afacan, Luis A. Camuñas-Mesa, Bernabé Linares-Barranco, and Haralampos-G. Stratigopoulos. 2021. Neuron fault tolerance in spiking neural networks. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 743–748.

[45] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* 32 (2012), 323–332.

[46] Michael B. Sullivan, Siva Kumar Sastry Hari, Brian Zimmer, Timothy Tsai, and Stephen W Keckler. 2018. Swapcodes: Error codes for hardware-software cooperative gpu pipeline error detection. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 762–774.

[47] Michael B. Sullivan, Nirmal Saxena, Mike O'Connor, Donghyuk Lee, Paul Racunas, Saurabh Hukerikar, Timothy Tsai, Siva Kumar Sastry Hari, and Stephen W. Keckler. 2021. Characterizing and mitigating soft errors in gpu dram. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 641–653.

[48] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. 2019. The impact of GPU DVFS on the energy and performance of deep learning: An empirical study. In *Proceedings of the 10th ACM International Conference on Future Energy Systems*. 315–325.

[49] Haoxin Tu, Lingxiao Jiang, Jiaqi Hong, Xuhua Ding, and He Jiang. 2024. Concretely mapped symbolic memory locations for memory error detection. In *IEEE Transactions on Software Engineering* 50, 7 (2024), 1747–1767.

[50] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747. Retrieved from https://arxiv.org/abs/1708.07747

[51] Dawen Xu, Cheng Chu, Qianlong Wang, Cheng Liu, Ying Wang, Lei Zhang, Huaguo Liang, and Kwang-Ting Cheng. 2020. A hybrid computing architecture for fault-tolerant deep learning accelerators. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 478–485.

[52] Geng Yuan, Zhiheng Liao, Xiaolong Ma, Yuxuan Cai, Zhenglun Kong, Xuan Shen, Jingyan Fu, Zhengang Li, Chengming Zhang, Hongwu Peng. 2021. Improving DNN fault tolerance using weight pruning and differential crossbar mapping for reram-based edge AI. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 135–141.

[53] Jeff Jun Zhang, Kanad Basu, and Siddharth Garg. 2019. Fault-tolerant systolic array based accelerators for deep neural network execution. *IEEE Design & Test* 36, 5 (2019), 44–53.

[54] Wei Zhao and Yu Cao. 2006. New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Transactions on Electron Devices* 53, 11 (2006), 2816–2823.

Received 8 September 2023; revised 28 August 2024; accepted 10 October 2024