ZOR: Zero Overhead Reliability Strategies for AI Accelerators

ZOR: Zero Overhead Reliability Strategies for AI Accelerators

Eleonora Vacca, Sarah Azimi, Luca Sterpone
Politecnico di Torino, Turin, Italy
{eleonora.vacca, sarah.azimi, luca.sterpone}@polito.it

***Abstract*— This research investigates the crucial integration of Neural Network (NN) models with the architecture of the hardware (HW) accelerator. Unlike existing approaches overlooking this interaction, we emphasize understanding the accelerator Datapath for reliability-focused algorithmic solutions. Focusing on Systolic Arrays Datapath, we theoretically evaluate the fault propagation from the HW layer to the NN. This analysis identifies variations in fault effects linked to various data mapping strategies. Considering the fault propagation model, we propose a novel reliability-oriented mapping strategy to mitigate fault effects based on resource rotation. Validation through HW fault injection demonstrates that an architecture-aware NN implementation reduces the impact of faults by up to 40%. Moreover, experimental results indicate that our proposed solution enhances the NN resilience, resulting in up to a 30% reduction in the error rate. Most importantly, these enhancements are attained without introducing performance or hardware overhead.**

Keywords—CNN, Algorithm, Systolic Array, Reliability, FPGA

I. INTRODUCTION

Progress in Deep Learning[1] methods is focused on elevating the efficacy of neural networks (NN), addressing key performance metrics such as inference accuracy, latency, computational load, and energy consumption. This pursuit involves refining sophisticated models and innovating accelerator architectures dedicated to supporting the computational demands and enhancing the parameters mentioned. Nevertheless, the widespread integration of NN models in safety-critical domains such as autonomous driving, medicine, and avionics underscores the equal importance of reliability alongside performance due to the severe implications in human life.

Despite the numerous reliability analyses conducted separately on NN models and hardware (HW) accelerators in the existing literature [2], there is often a lack of comprehensive assessments considering both aspects due to the wide gap between the physical hardware characteristics and the model behavior. Reliability evaluations of NN models often overlook the underlying HW layer, focusing on methodologies such as NN's weight data corruption [3] to assess the model's inherent robustness. Likewise, when assessing reliability in the accelerators, attention is typically directed toward the high-level consequences of faults in the Datapath [4], neglecting a comprehensive understanding of why a specific fault resulted in a particular behavior in the NN. This consideration gains particular significance when exploring the execution of NN on diverse HW architectures, as the circuit topology significantly shapes the NN's response to HW faults. Furthermore, within an accelerator Datapath, it is crucial to acknowledge that translating the NN model into machine instructions and the execution flow may not be univocal. While the ultimate execution result remains consistent, the translations can exhibit variations in both performance and reliability. Hence, it is essential to consider a hardware-oriented formulation of algorithmic implementations to avoid adopting suboptimal, this imperfect, solutions. Indeed, typical approaches to enhance system reliability rely on adopting redundancy. This involves allocating multiple computation nodes that perform the same calculations in parallel, whether aligned in time or not, and combining the results through majority voting. In the realm of NN, the concept of redundancy extends to neurons. Given that NNs consist of thousands of neurons, replicating the computations of each neuron is impractical. Hence, some authors proposed applying redundancy techniques, such as triple modular redundancy, only to neurons identified as critical [5][6]. However, it is worth noting that introducing redundancy, whether in terms of resources or computation, incurs a non-negligible cost.

This work highlights that a thorough understanding of the Datapath alone can drive algorithmic decisions, enhancing reliability without introducing additional costs, whether in hardware resources or performance.

We used an open-source Tensor Processing Unit (TPU) as a Systolic Array (SA) accelerator. Initially, we analyzed the circuit topology to evaluate how Convolutional Neural Networks (CNNs) are mapped to the core. Next, leveraging the binding between HW resources and higher-level computations, we evaluated how the errors induced by faults in the Datapath propagate and their further consequences. In particular, we conducted analytical comparisons to evaluate the impact of faults under different data mapping policies applied to CNN. From the theoretical analysis, we concluded that employing the Input Stationary (IS) strategy in the SA Datapath enhances overall robustness. Then, drawing on the fault propagation model, we proposed a novel resource-rotation mapping strategy to improve the fault tolerance of the CNN when adopting the Weight Stationary (WS) policy. To validate our conclusions, we performed experiments, introducing hardware faults. We implemented the accelerator on an SRAM-based FPGA and adopted the emulation-based hardware fault injection. This involved inserting bit-flips within the device Configuration RAM (CRAM), allowing to emulate structural faults in the SA Datapath. All the tested algorithmic implementations were executed under the same fault conditions to draw a meaningful comparison and ensure a fair evaluation. We selected two CNN architectures implementing MNIST-digit and CIFAR10 classifications. Experimental results show that by carefully selecting the data mapping strategy, it is possible to achieve a 30% and 18% reduction in the fault-induced effects for the MNIST and CIRFAR10, respectively, without any modifications in the HW architecture. Furthermore, by adopting our resource rotation strategy, even the most sensitive mapping policy, the WS, can be enhanced, obtaining comparable results as for the IS case.

The paper is organized as follows. Section II presents an overview of previous works, while Section III introduces the functionalities of the SA accelerator. Section IV details the Software-to-Hardware Mapping Strategies, and Section V describes the proposed reliability assessment. Section VI concludes.

II. RELATED WORKS

As CNNs are increasingly used in safety-critical applications, assessing their reliability has gained significant attention. Numerous studies have been conducted to explore the effects of failures on NNs. Several fault-injection-based

approaches are available in the literature. Software-based fault injections such as TensorFI [7] and PyTorchFI [8] simulate errors within the NN by introducing faults as bit-flips in the weights and biases of the model. These approaches are hardware-independent, therefore they are referred to as an abstracted model without any realistic correlation with the physical hardware. In contrast, adopting hardware-level fault injection allows correlating DNN errors with the accelerators [9][10]. Indeed, as the circuit topology and technology vary, the fault-induced error in the NN inference process changes.

Although SAs are far from a new architectural proposal, dating back to the 80s [11], in the past decade, they have proven to be among the most efficient accelerators in terms of TBops/W [12]. Literature has revisited the topic, focusing on computational efficiency, developing new SA accelerators suited for state-of-the-art AI models [13][14], and reliability aspects [15]. The latter includes exposure to accelerated radiation tests to induce Single Event Effects [16][17] and Datapath fault simulation through netlist corruption in HDL simulation environments [18]-[20]. SRAM-based FPGAs are also utilized to evaluate the reliability of AI models through hardware fault emulation [21]-[24]. SAs featuring multiple processing elements (PEs) challenge conventional hardware redundancy-based hardening methods. Time redundancy, although an option, may introduce inference execution delays [25], potentially compromising the ability to meet real-time response constraints. Once the fault location is identified [28][29], approaches such as NN model compression and weight pruning are employed [26][27]. However, no solution is currently available for improving the reliability without necessitating interventions in both the HW and the NN model. In our proposed approach, exploiting knowledge of circuit topology, data movement, and NN structure, we implement algorithmic strategies to enhance reliability. The effectiveness of the proposed methods is validated through hardware-fault injection experiments, illustrating that mitigating fault effects is achievable through architecture-aware formulation of algorithms without imposing performance penalties or resource overhead.

III. Background on SA

SAs-based accelerators are gaining popularity thanks to their ability to provide performance and energy efficiency [12]. The computational core is organized as a 2D array of processing elements (PEs), interconnected through fixed data paths designed to facilitate efficient data and computation flow. Specifically, connections between PEs in the same column transmit intermediate computation results, while connections between PEs in the same row transfer inputs (or weights). Each PE computes a multiplication between an input value ($x_{i,j}$) and a corresponding weight ($w_{i,j}$) and accumulates the result in a single clock cycle. The SA Datapath may be equipped with external accumulators to accommodate operation tiling when the size of the data matrices does not fit in the available resources. To execute a complete NN inference on SA, each layer has to be converted into a General Matrix Multiplication (GEMM) operation [32]. The conversion is straightforward in the case of a fully connected layer, while it requires some data processing when dealing with the convolutional layer. However, once translated, there are several ways to map the operation in the hardware resource of the Datapath, each providing different performance and reliability, as we will demonstrate in the following.

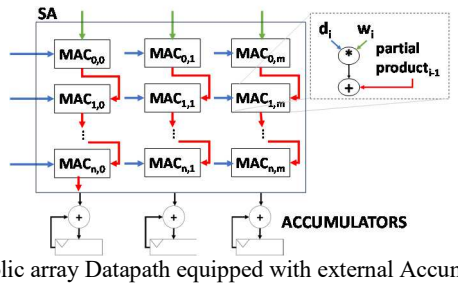

Fig. 1 Systolic array Datapath equipped with external Accumulators for operation tiling.

IV. Software-to-Hardware Mapping Strategies in SA

The data mapping policies for SAs dictate the strategies employed in allocating and distributing computations within this parallel computing architecture. Three primary mapping policies exist: Weight Stationary (WS), Input Stationary (IS), and Output Stationary (OS). In WS, the weight matrix is loaded in the PEs grid before computation starts with a 1:1 correspondence between the data matrix and the SAs, while the activations flow from the left to the right of the array. Similarly, the activation matrix is loaded first in the IS case, and the weights data spans all the columns. Finally, in the OS policy, the activation data moves to the right PE, and the weight data moves to the lower PE. While WS and OS policies have been investigated[30], previous works have not addressed IS mapping. Moreover, the authors' evaluations were performed in a time-consuming simulation environment, limiting the benchmark to a small NN layer. In contrast, we compare WS and IS starting from a theoretical approach, moving to hardware execution of entire NN models.

A. CNN Mapping Strategies

CNNs are characterized by multiple convolutional layers, each comprising diverse filters designed for feature extraction. Each filter consists of small weight matrices, the kernels, whose number depends on the number of channels in the input image. For example, consider an RGB image with dimensions (N, M, 3) and a convolutional layer featuring 128 filters. Since the input matrix has three channels, each f_i filter comprises three kernels with dimensions (n, m). Each channel kernel will convolve with the related channel input matrix. Then, the three channel-wise convolutions result is accumulated, producing a single output feature map for f_i. Assuming a stride of s, the overall result of the convolutional layer will have dimension ((N-n)/s, (M-m)/s, 128). Hence, the produced output, input for the next layer, consists of 128 channels. Consequently, as we delve into the hidden layers of the CNN, the layer weight matrix's complexity depends on the number of layer filters F_i and the F_{i-1}, defining the number of kernels per filter. Meanwhile, the input size decreases due to the stride. This implies that WS and IS policy will likely show distinct performance characteristics, as one might anticipate. However, beyond performance considerations, these two mapping policies also carry implications for reliability. To identify which strategy is more reliable, it is essential to explore how the operations described earlier are mapped within the SA. Consider the illustrated RGB case per simplicity, with F filters in the convolutional layer. Each filter f_i is defined by (k^i_R, k^i_G, k^i_B). As each filter has to convolve with the R, G, and B input matrices, if the process is mapped with no optimization to the SA, this would imply reading the same input matrices F times. Moreover, due to the SA interconnection path characterized by propagating the data from left to right of the array, shared among PE columns, if WS policy is adopted, then to preserve the mathematical consistency for each f_i filter, each k^i convolution should be executed separately from that of the other channels. This results in only n*m PEs processing the data. Adopting the IS

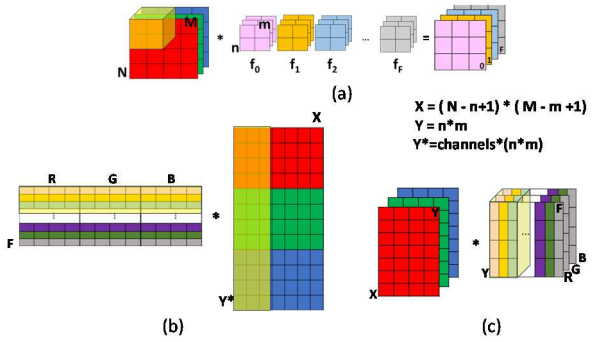

Fig. 2 Convolutional layer (a) high-level model of (b) GEMM-based in [31], (c) our proposed implementation.

policy increases the number of resources involved in computation. Still, the required input matrix read operations are suboptimal since the same input data needs to be loaded in the SA F times. To reduce this reading overhead, the computation should be organized as GEMM. Drawing on what was proposed in [31], we implemented custom solutions tailored for SA equipped with external accumulators.

1) Channel-wise GEMM Implementation

Kernels related to the same channel but belonging to different filters have to be multiplied for the same input matrix. Considering the RGB case, three matrices, one per channel, are realized: W_R, W_G, and W_B. Each matrix has size (n*m, F) and is constructed such that each column, indexed by i, corresponds to the vectorized channel kernel of f_i, as shown in Fig. 2c. In contrast, in [31], one filter matrix F_m of size (F, C*(n*m)) is realized, with C number of channels.

In our proposed solution, each channel input matrix I_C (R, G, and B in the example) goes through the Img2row transformation, which consists of constructing a bigger matrix, where row content is the flattened 2D convolution window. Consequently, the number of rows, X, reflects the number of windows necessary to convolve the entire matrix with a given stride. Similarly, in [31], the D_m matrix is constructed by applying the same window unfolding operation to create columns and stacking the channel matrices, as in Fig 2b. Following [31] implementation, to accommodate the operation on the SA, D_m and F_m are partitioned as follows:

$$D^i = D_m[\ C*(n*m), i : i + SA_{size},]\ \ with\ i \in [0, \frac{X}{SA_{size}}[\quad (1)$$

$$W^j = F_m[\ j : j + SA_{size}, C*(n*m)]\ \ with\ j \in [0, \frac{F}{SA_{size}}[\quad (2)$$

Then each partition is again divided into square submatrices to exactly match the SA size, resulting in

$$D^i = [d_0, d_1 \dots d_P]\ \ W^j = [w_0, w_1 \dots w_P]\ \ with\ p\ \in [0, P[\quad (3)$$

where P = C*(n*m)/SA_{size}. To perform convolution, each submatrices set D^i is processed as follows:

$$\sum_{p=0}^{P} w_p^j * {d_p}^i \quad with\ {d^i}_p \in D^i,\ \ w_p^j \in W^j \quad (4)$$

The multiply and accumulate process in (4) has to be repeated for all the D^i considering the same W^j and then again for all the weight partitions W^j. This results in the same weight matrices w_p^j being loaded X/SA_{size}, and the same ${d^i}_p$ F/ SA_{size} times. Our proposed solution aims to reduce the number of weights loading. In our solution, the channel matrix I_C is multiplied by its W_c, and the channels' outputs are merged using the external Accumulators while the SA is processing next-channel computation. In this scenario, each W_c is partitioned in F/ SA_{size} square matrices of SA_{size}. Then, each square submatrix is loaded just once, while I_C is processed with no partitioning, i.e., spanning all rows from 0 to X. With this approach, we reduce the loading operations of each weight submatrix from X/SA_{size} times to 1. In the following, we will compare the reliability of this proposed GEMM implementation when adopted with WS and IS.

V. Reliability Evaluation of Mapping Strategies

To assess the reliability implications of WS and IS mapping strategies, it is crucial to correlate the computations performed by the SA with their meaning in the CNN architecture. In contrast to previous studies that evaluated the reliability starting from fault injection campaigns [30] we first propose an analytical approach to model the fault-induced error propagation from the PEs grid to the NN model. Analyzing the propagation model, we will identify and propose reliable algorithmic solutions. Then, we will exploit the hardware fault injection to validate our findings.

A. Fault-induced Error Propagation

Previously, we explained how the convolutional layer is mapped on the SA. Considering the scenario in Fig. 3, where a fault is located in a PE. Due to per-column accumulation, the faulty partial product will flow along the column and accumulate with those produced by the other PEs, producing, in the end, a faulty output. This behavior is independent of the mapping policy since it relates to circuit topology and hence holds both for the [31] implementation and for our channel-wise GEMM. On the other hand, the consequences of the faulty output differ between WS and IS policies.

The WS policy is characterized by the 1:1 correspondence between the PEs grid and the weight matrix. Hence, the computations of filter f_i (for each of its channels) are always mapped to column i. Each row vector in the input matrix is the image window to be convolved with. Given that, the multiplication of row_j by f_i produces one pixel $p^j{}_i$ of the *i-th* output feature map.

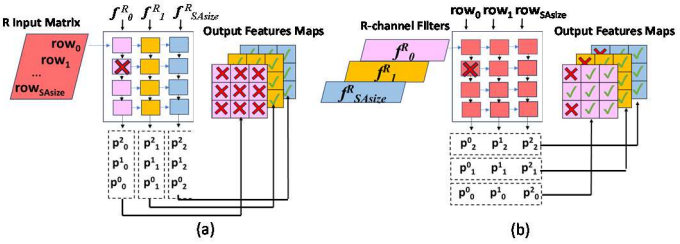

Fig. 3. CNN fault propagation with (a) WS and (b) IS mapping.

A fault affecting the f_i computation impacts all pixels of its output, as shown in Fig. 3a, while the other filters' computations are preserved. This behavior is the same as for [31] convolution implementation, as found in the experiment conducted by[30]. In the IS policy, the behavior is inverted with the image rows mapped statically, one in each column of the SA, while filters' weights flow among the PEs. Hence, considering the same fault scenario, the faulty column is responsible for computing the output pixels for all the filters, resulting in faulty pixels affecting all the output features map, located in the same position, as shown in Fig. 3b. The number of faulty pixels strictly depends on the SA size. Due to operation tiling, the computation is organized in blocks of SA_{size} rows loaded in the PEs grid. Therefore, one faulty PE will produce one corrupted pixel every SA_{size}, for all the output feature maps. Going deeper into the CNN hidden layers, the feature maps are reduced in size, decreasing the reuse of faulty resources per layer. In the WS case, since the operation tiling occurs on the weight matrix, a faulty column will result in faulty output feature maps (i.e, not just few pixels) every SA_{size} output maps. The severity of the error,

both in the IS case and WS depends on fault location, as faults in LSBs are typically less critical, and other mechanisms, like rounding during accumulation or data values, may mask the fault effects. However, completely failing in extracting features, as for the WS, has a higher impact on CNN accuracy, as opposed to having a few faulty pixels in every feature map that may be interpreted as noise.

To mitigate this effect, we propose a mapping strategy that forces resource rotation through data. Reminding that in WS, each kernel $k^i{}_c$ related to a filter f_i is always mapped in the same column i of resources, we arranged the weights matrices such that for each consecutive weight loading in the SA, the data columns are shifted and rotated. In detail, considering our channel-wise GEMM convolution with C channels. At each iteration c, with $c \in C$, the weights of $k^i{}_c$ are mapped to the SA column $((i+c) \bmod SA_{size})$. Therefore, we force the execution of filter f_i channels computation over different resources. As a result, just one channel output is faulty every SA_{size} channel processed, instead of all as for canonical WS. The channel outputs are then accumulated, with the external accumulators, to produce a single output future map for the filter f_i, as required by the convolution algorithm. The impact on the final output feature map depends on (i) the number of channels, with higher the number lower the impact during the accumulation (i.e., in RGB case, one faulty channel output is likely sufficient to impact the result concerning one faulty channel over 64) (ii) SA size, which determines how many times the same column is used to process several channels for the same f_i. However, by forcing rotation, also the other filters' computations will be affected by the same faulty behavior, while in canonical WS their computation is preserved. Still, spreading a few corrupted channel computations over all the filters, whose impact is smoothed by the accumulation process, is more likely to introduce noise in the NN rather than errors.

B. Validation through Fault Injection

To validate the theoretical approach characterizing the reliability of the SA accelerator related to the different algorithmic solutions, an open-source TPU [33] accelerator has been implemented on a Xilinx Zynq 7020 SoC equipped with SRAM-based FPGA, where the accelerator was mapped. Following state-of-the-art implementation, the SA's PEs are mapped on the on-chip DSP available in the programmable logic[34]. The SA size is 14 x 14 PEs, exploiting all the available resources, and runs at 177MHz. The open-source accelerator comes with a framework capable of translating only fully connected NN into the core's Instruction Set Architecture. Consequently, we implemented a framework that, starting from a high-level CNN, translates the NN into a sequence of elementary assembly instructions, according to the mapping strategy adopted. Since on HW execution, in contrast with[30] that limited their evaluation to single layers, we evaluated two complete CNN models targeting MNIST-digit and CIFAR10 datasets. Details about the CNNs, implemented with QKeras [35] are in Table 1.

Table 1. CNN models characteristics.

	Number of Conv Layer	Number of FC Layer	Tot. Parameters	Inference time [ms]	
				[31]	C-wise
MNIST	3	1	40,874	0.134	0.129
CIFAR10	6	1	91,648	1.8	1.2

To evaluate the reliability, we utilized the HW fault emulation. FPGA designs are implemented through bitstream, whose content configures the device resources (logic, interconnection, and memory) to implement the target circuit. Hence, by manipulating the bitstream, it is possible to model structural faults[36][37] in the Datapath.

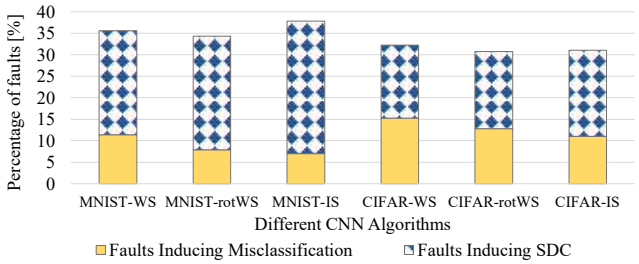

Fig. 4. Experimental results over 5,000 faults injected.

The injection campaign comprised 5,000 fault injections. Each injected fault affected only one PE at a time. For each evaluated fault, both MNIST-digit and CIFAR10 classifications were executed. Therefore, the two CNNs were evaluated under the same fault conditions, first with WS policy and then with IS. Please consider that the bitstream upload requires a few seconds, while for each evaluated fault, the NN weights need to be sent to the core. These operations are time-consuming. Since our goal is to evaluate fault effects under different algorithms, we focused on performing more fault injections rather than classification per CNN, limiting to 10 images per fault per CNN (accuracy degradation is deeply investigated in previous works). The results, in Fig. 4, are prosed as the percentage of injected faults that induced misclassifications, and as the percentage of fault inducing Silent Data Corruption (SDC). As SDC, we considered variation in the top-1 class score, which did not result in the wrong classification. The results demonstrate the findings of our theoretical approach, highlighting that the same CNN architecture shows greater resilience when mapped in IS policy, showing a reduction in faults causing misclassification of 30% for MNIST and 18% for CIFAR10. Indeed, IS is characterized by a higher percentage of SDC, implying that the fault has induced effects in the computation, which deviates from the golden reference but ensures correct classification. On the other hand, the higher the complexity of the CNN architecture, the higher the sensitivity to the same faults. Indeed, results show that a fault whose effect is masked in the lighter CNN (MNIST) is provoking a pattern of misclassification in the deeper CNN (CIFAR10). Additionally, as CIFAR10 is based on RGB images, each layer's complexity is higher than that of MNIST, which translates into the fact that the faulty unit is used multiple times to extract multiple features, wrongly. On the same faulty bitstreams, we evaluated our proposed resource rotation algorithm. Results show a reduction in the impact of faults of 30.57% in the MNIST and 16.01% in CIFAR10 considering the canonical WS approach, demonstrating that resource rotation could be an effective, low-cost solution to improve the reliability of the SA accelerator further. These prominent results encourage us to keep investigating this aspect in the future.

VI. Conclusions

In this paper, we studied SA Datapath and CNN mapping strategies to identify the high-level fault effects. Our analysis targeted two canonical mapping strategies and favored the IS strategy over WS for reliability. We also proposed a mapping strategy based on data-driven resource rotation. All the theoretical analyses have been evaluated on HW by implementing SA on SRAM-based FPGA. To validate our findings, we performed HW fault injection, which not only confirmed our approach but also suggested that zero overhead reliability strategies are practical solutions. Future work will explore automating the resource rotation mechanism within the SA architecture.

REFERENCES

[1] Y. LeCun et al., “Deep learning,” Nature, vol. 521, May 2015, Art. no. 436, doi: 10.1038/nature14539.

[2] A. Bosio et al., "A Reliability Analysis of a Deep Neural Network," 2019 IEEE Latin American Test Symposium (LATS), Santiago, Chile, 2019, pp. 1-6, doi: 10.1109/LATW.2019.8704548

[3] G. Gavarini et al., "SCI-FI: a Smart, aCcurate and unIntrusive Fault-Injector for Deep Neural Networks," 2023 IEEE European Test Symposium (ETS), Venezia, Italy, 2023, pp. 1-6, doi: 10.1109/ETS56758.2023.1017395

[4] Y. Ibrahim et al., "Soft errors in dnn accelerators: A comprehensive review", Microelectronics Reliability, vol. 115, pp. 113969, 2020.

[5] T. G. Bertoa et al., "Fault-Tolerant Neural Network Accelerators With Selective TMR," in IEEE Design & Test, vol. 40, no. 2, pp. 67-74, April 2023, doi: 10.1109/MDAT.2022.3174181.

[6] A. Ruospo et. al, "Selective Hardening of Critical Neurons in Deep Neural Networks," 2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Prague, Czech Republic, 2022, pp. 136-141, doi: 10.1109/DDECS54261.2022.9770168.

[7] Z. Chen et al.,"TensorFI: A flexible fault injection framework for TensorFlow applications", 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), pp. 426-435, 2020, October.

[8] A. Mahmoud et al., "PyTorchFI: A Runtime Perturbation Tool for DNNs," 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Valencia, Spain, 2020, pp. 25-31, doi: 10.1109/DSN-W50199.2020.00014.

[9] F. F. d. Santos et al., "Analyzing and Increasing the Reliability of Convolutional Neural Networks on GPUs," in IEEE Transactions on Reliability, vol. 68, no. 2, pp. 663-677, June 2019, doi: 10.1109/TR.2018.2878387.

[10] M. Taheri et al., "DeepAxe: A Framework for Exploration of Approximation and Reliability Trade-offs in DNN Accelerators," 2023 24th International Symposium on Quality Electronic Design (ISQED), San Francisco, CA, USA, 2023, pp. 1-8, doi: 10.1109/ISQED57927.2023.1012935

[11] D. I. Moldovan, "On the design of algorithms for VLSI systolic arrays," in Proceedings of the IEEE, vol. 71, no. 1, pp. 113-120, Jan. 1983, doi: 10.1109/PROC.1983.12532.

[12] B.Peccerillo et al."A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives", Journal of Systems Architecture, Volume129,2022,102561,doi: 10.1016/j.sysarc.2022.102561.

[13] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 1-12 , doi: 10.1145/3079856.3080246.

[14] H. -Y. Wang et al.,, "Row-wise Accelerator for Vision Transformer," 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Incheon, Korea, Republic of, 2022, pp. 399-402, doi: 10.1109/AICAS54282.2022.9869928.

[15] E. Vacca et al., "A Comprehensive Analysis of Transient Errors on Systolic Arrays," 2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Tallinn, Estonia, 2023, pp. 175-180, doi: 10.1109/DDECS57882.2023.10139763.

[16] R. L. R. Junior and P. Rech, "Reliability of Google’s Tensor Processing Units for Convolutional Neural Networks," 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S), Baltimore, MD, USA, 2022, pp. 25-27, doi: 10.1109/DSN-S54099.2022.00018.

[17] D. P. Ramaswami et al "Single Event Upset Characterization of the Intel Movidius Myriad X VPU and Google Edge TPU Accelerators Using Proton Irradiation," 2022 IEEE Radiation Effects Data Workshop (REDW) (in conjunction with 2022 NSREC), Provo, UT, USA, 2022, pp. 1-3, doi: 10.1109/REDW56037.2022.9921608.

[18] Kundu et al., "Toward Functional Safety of Systolic Array-Based Deep Learning Hardware Accelerators," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 3, pp. 485-498, March 2021, doi: 10.1109/TVLSI.2020.3048829.

[19] J. J. Zhang et al., "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," 2018 IEEE 36th VLSI Test Symposium (VTS), San Francisco, CA, USA, 2018, pp. 1-6, doi: 10.1109/VTS.2018.8368656.

[20] A. Siddique et al., "Exposing Reliability Degradation and Mitigation in Approximate DNNs Under Permanent Faults," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, doi: 10.1109/TVLSI.2023.3238907.

[21] De Sio et al., "FireNN: Neural Networks Reliability Evaluation on Hybrid Platforms," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 2, pp. 549-563, 1 April-June 2022, doi: 10.1109/TETC.2022.3152668.

[22] B. Du et al., "On the Reliability of Convolutional Neural Network Implementation on SRAM-based FPGA," IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2019, pp. 1-6.

[23] E. Vacca et al., “Analyzing the SEU-induced Error Propagation in Systolic Array on SRAM-based FPGA”, IEEE Radiation and its Effects on Components and Systems (RADECS), 2023.

[24] X. Dawen et al., "Reliability Evaluation and Analysis of FPGA-Based Large Scale Integration (VLSI) Systems, vol. 29, no. 3, pp. 472-484, March 2021, doi: 10.1109/TVLSI.2020.3046075.

[25] A. Antola et al., "Policies for fault-tolerance through mixed space- and time-redundancy in semi-systolic FFT arrays," doi: 10.1109/ARRAYS.1988.18093.

[26] K. T. Chitty-Venkata et al., "Model Compression on Faulty Array-based Neural Network Accelerator," 2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC), Perth, WA, Australia, 2020, pp. 90-99, doi: 10.1109/PRDC50213.2020.00020.

[27] J. J. Zhang et al.,"Fault-Tolerant Systolic Array Based Accelerators for Deep Neural Network Execution," in IEEE Design & Test, vol. 36, no. 5, pp. 44-53, Oct. 2019, doi: 10.1109/MDAT.2019.2915656.

[28] E. Vacca et al., "RunSAFER: A Novel Runtime Fault Detection Approach for Systolic Array Accelerators," 2023 IEEE 41st International Conference on Computer Design (ICCD), Washington, DC, USA, 2023, pp. 596-604, doi: 10.1109/ICCD58817.2023.00095.

[29] H. Lee et al.,"STRAIT: Self-Test and Self-Recovery for AI Accelerator," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, doi: 10.1109/TCAD.2023.3236875.

[30] U. K. Agarwal et. al, "Towards Reliability Assessment of Systolic Arrays against Stuck-at Faults," 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, doi: 10.1109/DSN-S58398.2023.00063.

[31] Sharan Chetlur et al., "cudnn: Efficient primitives for deep learning", arXiv preprint arXiv:1410.0759, 2014.

[32] A. Anderson et al., "High-Performance Low-Memory Lowering: GEMM-based Algorithms for DNN Convolution," 2020 IEEE 32nd International Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD), Porto, Portugal, 2020, pp. 99-106, doi: 10.1109/SBAC-PAD49847.2020.00024.

[33] Jonas Fuhrmann, “ Implementierung einer Tensor Processing Unit mit dem Fokus auf Embedded Systems und das Internet of Things”, 2018, http://hdl.handle.net/20.500.12738/8527

[34] Xilinx. (2017) Deep Learning with INT8 Optimization on Xilinx Devices. [Online]

[35] Claudionor N. Coelho Jr et al., "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors", Nature Machine Intelligence (2021).

[36] E. Sanchez et al."Effective emulation of permanent faults in ASICs through dynamically reconfigurable FPGAs," 24th International Conference on Field Programmable Logic and Applications (FPL), 2014, pp. 1-6.

[37] L. Bozzoli, et al., "PyXEL: An Integrated Environment for the Analysis of Fault Effects in SRAM-Based FPGA Routing," International Symposium on Rapid System Prototyping (RSP), pp. 70-75, 2018, Turin, Italy, 2018, pp. 70-75, doi: 10.1109/RSP.2018.8632.