Towards Reliability Assessment of Systolic Arrays against Stuck-at Faults

Udit Kumar Agarwal, Abraham Chan, Ali Asgari, Karthik Pattabiraman
Electrical and Computer Engineering, University Of British Columbia, Vancouver, Canada
uditagarwal1997@gmail.com, abrahamc@ece.ubc.ca, aliasgarikh97@gmail.com, karthikp@ece.ubc.ca

***Abstract*—Neural Networks are ubiquitously used in safety-critical applications such as autonomous vehicles and medical diagnostics. The increasing complexity and compute-intensiveness of deep neural networks (DNN) have motivated the need for DNN accelerators like Google's Tensor Processing Unit (TPU) to accelerate convolution and matrix multiplication operations. At its core, a TPU consists of a 2-Dimensional array of Multiply and Accumulation Units, called a systolic array, which is susceptible to permanent (e.g., stuck-at faults in the data path) and transient hardware faults (e.g., radiation-induced).**

We propose an RTL-level fault injection (FI) framework for systolic arrays. Using this framework, we characterize the software effect of errors (called Fault Patterns) induced by stuck-at faults within the multiply and accumulation units of the systolic array. We further analyze the effect of different data flows mapping schemes (output and weight stationery), operation types (convolution and matrix multiplication), and operation configurations (e.g., input size, convolution kernel size). Through the FI experiments, we categorized the fault patterns for stuck-at faults into well-defined classes based on their spatial patterns.

***Index Terms*—Systolic Array, Stuck-at Faults, Fault Injection**

I. Introduction

State-of-the-art deep neural networks (DNNs), such as GPT-3 [27] and BeRT [25], have millions of parameters and are computationally intensive. DNNs consist of multiple layers, out of which fully-connected layers (implemented as matrix multiplication) and convolution layers are the most expensive ones, with the time complexity of $O(Height * Width * Channels)$ where Height, Width, and Channels are the dimensions of the matrices being multiplied. To improve energy efficiency and reduce the execution overheads of training and inference of DNNs, DNN accelerators like Google's Tensor Processing Units (TPU) [12] and Microsoft's BrainWave [7] use a 2-D array of multiply and accumulation (MAC) units, called a systolic array, at their core for batch multiplication and addition operations.

Systolic arrays are designed to take advantage of the parallelism inherent in DNN computations. By dividing the input data into smaller chunks and distributing the work across a grid of MAC units, systolic arrays can perform many computations simultaneously, which can significantly reduce the overall time required to complete a task. For instance, Google's TPU 1 consists of 65K MAC units arranged in a 256x256 2-Dimensional array; TPU 3 and TPU 4 use multiple 128x128 systolic arrays. Google TPU 1 can achieve upto 92 TOPS [12], while Google TPU 3 and TPU 4 can deliver upto 123 TFLOPS (Trillion Floating Point Operations Per Second) and 275 TFLOPS, respectively [1]. These numbers are typically much higher than the corresponding TFLOPs obtained on CPUs and GPUs, and hence systolic arrays are becoming increasingly popular for DNN applications. However, the reliability implications of using systolic arrays have not been studied in depth.

Safety-critical applications such as autonomous driving systems (ADS) often use systolic arrays (like in EdgeTPU) to accelerate DNN inference. The ISO 26262 standard for functional safety of ADS requires that, for Automotive Safety Integrity Level D (ASIL-D), there should be no more than 10 hardware faults (both transient and permanent) in a billion hours of operation [22]. Hardware faults can be permanent (e.g., stuck-at fault in the data path of a MAC unit) or transient in nature (e.g., bit-flips in the data path). Permanent faults in systolic arrays can be caused by a variety of factors, including physical damage to the hardware, defects in the manufacturing process, and wear and tear over time. Transient faults, on the other hand, are typically induced due to high energy particles such as neutrons and alpha particles striking the hardware.

In this paper, we focus on permanent faults in systolic arrays. Unlike CPUs, systolic arrays are highly parallelized and reuse intermediate computation data; therefore, our hypothesis is that permanent faults in systolic arrays could lead to multiple software-level faults (i.e. multiple corruptions in tensor operators), which are referred to as *Fault Patterns*. This can significantly degrade DNN's accuracy. For instance, Zhang et al. [29] showed that the classification accuracy of CNN on the MNIST dataset [16] drops by 40% if even 0.01% (8 out of 65K) MAC units are affected by stuck-at faults.

Most prior work has focused on analyzing the effect of stuck-at faults on the DNN's accuracy [29]. However, it is not clear how these faults manifest at the intermediate layers of the DNNs. This is important because understanding fault manifestation at the intermediate layers not only provides insights into building more resilient DNN architectures [5], but also improves the accuracy of application-level FI tools. Existing application-level FI tools like TensorFI [18], PyTorchFI [19], and LLTFI [2] are much more scalable than RTL-level fault injection. However, these tools are restricted to CPU- and GPU- based models, and do not consider systolic arrays. Fault patterns at the intermediate layers of the DNN due to stuck-at faults in the systolic array can be used in tandem with application-level fault injectors to allow precise error modeling pertaining to the systolic array hardware in these tools.

In recent work, Rech et al. [23] classified the fault patterns

they observed for EdgeTPU via beam testing experiments. However, their classification is limited to transient faults, and not permanent faults. Further, they only consider the type, size of the tensor operation. However, other factors like the size of the systolic array itself, the type of data mapping scheme between the systolic array and the weights of the tensor operator (output and weight stationary) can also affect the observed fault patterns.

In this work, we extend Rech el al.'s [23] study to permanent faults, by characterizing the spatial attributes of software-level errors, referred to as fault patterns, introduced by the stuck-at faults in the MAC units of the systolic array. Understanding and classifying fault patterns is challenging, primarily because of the enormous state space for fault injection and also due to the masking of fault patterns by multiplication by near-zero DNN weights. We addressed these challenges by state-space sampling, and using pre-defined weight matrices, respectively.

As an initial step, we focus on the inference phrase of the DNN as opposed to the training phase. We further analyze how the induced fault patterns change with (1) different types of tensor operations (e.g., convolution, matrix multiplication), (2) the different mappings between the systolic array and the weights of the tensor operator (e.g., output and weight stationary), and (3) the input size of tensor operators.

Through extensive FI experiments at the RTL level using FPGAs, we found the fault patterns to be well-defined and further categorized them into classes based on the spatial distribution of faults. Our classification of fault patterns can be used along with existing application-level FI tools to accurately consider the systolic array hardware model.

II. Background

A. Type of DNN operations and their parameters

Deep Neural Networks comprise multiple operations like convolution, matrix multiplication (used for implementing 'dense' or 'fully-connected' layer), Relu, MaxPool, etc. Convolution and matrix multiplications, a.k.a generalized matrix multiplication (GEMM), are the most computation-intensive operations [14] and are thus accelerated using systolic arrays.

- **Fully-Connected Layer and GEMM**: Fully-connected (FC) layers with non-linear activation functions are used to learn a non-linear relation between the input features (extracted by the convolution layers) and the final prediction of the network. The FC layers are represented using matrix multiplication of input features and the neuron weights, followed by the addition of a bias term.
- **Convolution Layer**: Convolution layers are mainly used for extracting the features from the input image(s). The size of the extracted features depends on the kernel size, which is often quite small because the number of parameters in a convolution layer grows quadratically with the kernel size, according to the following equation: $NumParams = KernelSize^2 * InputChannel * OutputChannel$. Having a larger kernel size makes the CNN more computationally expensive without much benefit in accuracy. Moreover, convolution kernels used in DNNs consist of multiple output channels that are used to improve the robustness and accuracy of the DNN. For example, using multiple output channels to detect different features makes a DNN more resilient to input data variations, such as changes in ambient lighting.

In RQ2, we assess the effect of stuck-at faults for different types of operation, i.e., convolution and GEMM. Since different operations use the systolic array differently, we hypothesize that the fault patterns should vary for convolution and GEMM operations.

B. Implementing Convolution on Systolic Arrays

There are various methods for efficiently implementing convolution operations on systolic arrays. One of the most widely-used approaches is to convert the convolution into one large GEMM operation by:

- reshaping the input data ($N \times C \times H \times W$) into a 2-D matrix with dimensions $CRS \times NPQ$.
- reshaping the convolution kernel ($K \times R \times S$) into a 2-D matrix with dimensions $K \times CRS$.

Where N, C, H, and W are the batch size, number of channels, height, and width of the input data, respectively. Moreover, K, R, and S are the number of output channels, rows, and columns of the convolution kernel, while P and Q are the height and width of the output matrices. This implementation of convolution is used in CuDNN [6].

C. Operation Tiling

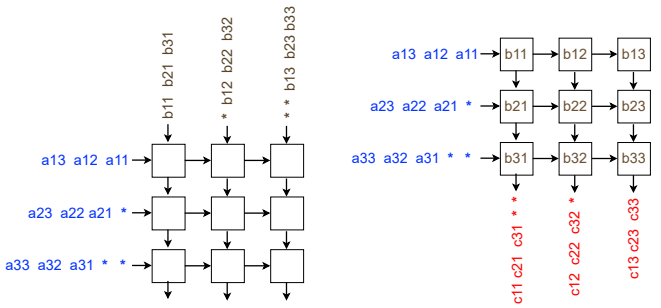
Operation tiling in a systolic array refers to the way in which a computation is divided into smaller sub-computations, or tiles, which are processed by the array in a parallel and pipelined manner. Operation tiling is useful when the computation is too large to be performed by a single processor. By dividing the computation into tiles, the systolic array can process the computation in a streaming fashion without having to store the entire computation in memory at once. This can be useful for tasks such as image and video processing, where the computation is often too large to fit into memory.

For instance, consider the matrix multiplication of two 4x4 matrices, A and B, as shown in Equation (1).

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} B = \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{bmatrix} \tag{1}$$

Assuming the systolic array size and, thus, the tile size to be 2×2. The matrices A and B are broken down into four tiles of 2×2 each, as shown in Equation (2) and Equation (3).

$$\begin{aligned} A_{tile1} &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad A_{tile2} = \begin{pmatrix} A_{13} & A_{14} \\ A_{23} & A_{24} \end{pmatrix} \\ A_{tile3} &= \begin{pmatrix} A_{31} & A_{32} \\ A_{41} & A_{42} \end{pmatrix} \quad A_{tile4} = \begin{pmatrix} A_{33} & A_{34} \\ A_{43} & A_{44} \end{pmatrix} \end{aligned} \tag{2}$$



(a) OS (b) WS

Fig. 1: OS (a) and WS (b) data flow schemes. In WS, the weights are stored within the systolic array, while in OS, partial outputs are stored in the systolic array.

The resultant matrix, C, is calculated through eight matrix multiplication and four matrix addition operations, as shown in Equation (4).

$$B_{tile1} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad B_{tile2} = \begin{pmatrix} B_{13} & B_{14} \\ B_{23} & B_{24} \end{pmatrix}$$
$$B_{tile3} = \begin{pmatrix} B_{31} & B_{32} \\ B_{41} & B_{42} \end{pmatrix} \quad B_{tile4} = \begin{pmatrix} B_{33} & B_{34} \\ B_{43} & B_{44} \end{pmatrix} \tag{3}$$

$$\begin{aligned} C_{tile1} &= A_{tile1} \times B_{tile1} + A_{tile2} \times B_{tile3} \\ C_{tile2} &= A_{tile1} \times B_{tile2} + A_{tile2} \times B_{tile4} \\ C_{tile3} &= A_{tile3} \times B_{tile1} + A_{tile4} \times B_{tile3} \\ C_{tile4} &= A_{tile3} \times B_{tile2} + A_{tile4} \times B_{tile4} \end{aligned} \tag{4}$$

D. Data flow Mapping Schemes

In a systolic array, data flow mapping refers to the way in which the data is routed between the MAC units of the systolic array. There are several different data flow mapping schemes, each with its own benefits and trade-offs.

Two common data flow mapping schemes are weight stationary (WS) and output stationary (OS). In WS data flow mapping, the weights of the convolution kernel are stationary, while the input data is moved between the MAC units, as shown in Figure 1b. This approach can be efficient for convolution operations with large kernel sizes, as it allows the convolution operation to be broken down into smaller steps that can be processed in parallel.

In OS data flow mapping, the output data is stationary, while the weights of the convolution kernel are moved between the MAC units, as shown in Figure 1a. This approach is efficient for convolutions with small kernel sizes, as it allows the entire convolution to be performed in a single step.

There are also other data flow mapping schemes that can be used in systolic arrays, such as input stationary and hybrid schemes that combine elements of both WS and OS mapping. The choice of data flow mapping scheme depends on the specific requirements of the application and the hardware architecture of the system.

E. Fault Model

In this work, we used the single stuck-at-fault model to evaluate the effect of permanent faults in the MAC units of the systolic array. Stuck-at faults in systolic arrays can be caused by a variety of factors, including physical damage to the hardware, defects in the manufacturing process, and wear and tear over time [30].

Among all the different permanent fault model, stuck-at fault model is fabrication technology and design style independent. Moreover, even though a single stuck-at fault does not accurately model some physical defects, the tests derived for single stuck-at faults are still valid for most defects including multiple stuck-at faults [21].

In addition, we make the following assumptions.

1) We do not consider faults in the memory elements as they can be protected with error correction codes (ECC).
2) We only consider faults in the data path as they can pass by silently and lead to wrong output without being detected [13].
3) We focus on faults in the MAC units as they make up most of the hardware area of the data path in the systolic arrays.

Our fault model is in line with prior work [15] [29] as well.

F. Fault Injection (FI) Framework

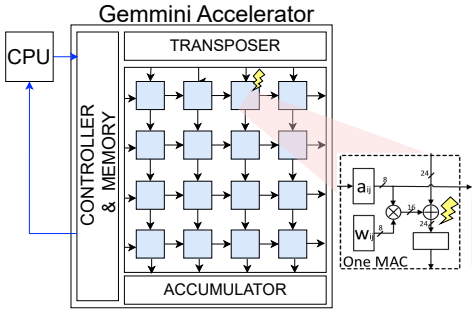


Fig. 2: Our simplified FI setup. To do FI, we inject a stuck-at fault in a randomly-selected MAC unit.

For this work, we use RTL-level fault injection because unlike Li et al. [18] that uses a pure-software level FI, RTL-level FI does not make any assumptions about the properties of the underlying hardware, thus making it more accurate. Moreover, unlike physical hardware-based FI [23], RTL-level FI is cost-effective as it does not require any special hardware.

Our RTL-based FI framework extends Gemmini [9] - a popular systolic array generator for evaluation of deep learning accelerator - to carry out FI in the processing elements. Gemmini is an open-source, full-stack DNN accelerator generator for DNN workloads, enabling end-to-end, full-stack implementation and evaluation of custom hardware accelerators. Figure 2 shows Gemmini's architecture and our fault injection setup. Along with the systolic array, Gemmini has hardware implementation of ReLU, DNN accelerator controller, scratchpad memory, and a single-core CPU called Rocket, to send

commands to the Gemmini DNN accelerator and (un)load data into the weight buffers.

We chose Gemmini because it is highly configurable: we can configure the size of the systolic array, data type, and type of data mapping scheme (OS vs. WS). The only limitation of Gemmini is scalability. Since it is a full-stack simulator, it takes considerable time to simulate a large DNN workload: several days to simulate the entire ResNet50 network.

For each FI experiment, we injected a single stuck-at fault in the intermediate signals of the MAC unit, right after the addition logic and before the result is stored in the accumulator. Unlike Zhang et al. [29] that injected multiple stuck-at faults (MSF), we chose to inject only a single stuck-at fault (SSF) because prior work has shown that the SSF fault model is sufficient to detect 98% of five or fewer MSF [3], [10].

III. Fault Injection Experiments

A. Research Questions and the Experimental Design

In this work, we aim to identify software-level fault patterns arising due to stuck-at faults in the MAC units of the systolic array. Toward this objective, we use an RTL-level FI framework (described in Section II-F) to simulate systolic arrays and do FI at runtime. Specifically, we ask the following research questions (RQs):

- How do the fault patterns change with different:
  (a) **RQ1:** data flow mapping schemes (OS and WS)? Are some data flows more fault tolerant?
  (b) **RQ2:** type of operations (convolution and GEMM)
  (c) **RQ3:** different size of operations, i.e., how does varying the size of GEMM or convolution affect fault patterns

There are two primary challenges involved in answering our RQs: first is the enormous state space, and second is the masking of faults due to multiplication by near-zero layer weights. We detail the challenges below.

Challenge 1: The fault patterns can be influenced by hardware configurations (like systolic array size, data mapping schemes, etc.), software configurations (operation type, operation configuration, size, etc.), and fault models (type of fault, fault location, fault bit position, etc.). This results in an enormous state space that is computationally expensive to explore in its entirety. For example, even a single systolic array of size 16×16, two data mapping schemes and two operation types and configurations, results in a state space with 131K different FI configurations. This is a conservative estimate.

To address this challenge, we *sampled the state space* and selected some configurations - based on their practicality and usefulness - while keeping other parameters constant. Table I shows the configuration settings for our RQs. Due to the scalability restrictions of Gemmini, we chose 16×16 as our systolic array size. Larger systolic array sizes require an impractically-large amount of system logic cells: 128×128 systolic array size requires ten times more system logic cells than available on our industrial-grade FPGA (Xilinx Virtex UltraScale+ VU9P). This scalability restriction is also common to prior work that uses the RTL model of TPUs [9]. For the input sizes (RQ3), we chose 16×16 and 112×112 input matrix sizes because the former is equal to the size of the systolic array (thus, no tiling effect), while the latter is larger than the size of the systolic array and we expect to observe the effect of tiling on the fault patterns.

TABLE I: Parameter configuration used for evaluating RQ1, and RQ2. The entries highlighted with brown are the ones whose effect we want to understand on the fault pattern.

	RQ1	RQ2	RQ3
Type of operation	GEMM	Conv vs GEMM	
Type of data mapping scheme	OS vs WS	WS	WS
Size of operation	16 x 16	16 x 16	16 x 16, 112 x 112
Size of Convolution Kernel	-	3x3x3x3, 3x3x3x8	
Size and Data Type of systolic array	16 x 16, INT8		

Similarly, for contrasting the effect of different tensor operators (GEMM and Conv) in RQ2, we chose two convolution kernel sizes, $3\times3\times3\times3$ and $3\times3\times3\times8$ ($R\times S\times C\times K$, refer Section II-B for notations). The former kernel size produces a 2-D matrix of size less than the size of the systolic array, while the latter kernel size produces a 2-D matrix of size more than the size of the systolic array, thereby resulting in a tiling effect in the convolution operator.

Challenge 2: It is quite common for weights of the DNN layers to be close to zero. These weights, when multiplied by the faulty computation (corrupted by the stuck-at fault), can suppress the fault pattern at the software level. In other words, the induced errors due to stuck-at faults might get masked due to multiplication by zero, thereby skewing the results of our RQs. To prevent this, instead of using real weights of the DNN layer, we used a uniform, non-zero weight matrix with all matrix elements set to one, to extract the fault pattern.

B. Evaluation Methodology

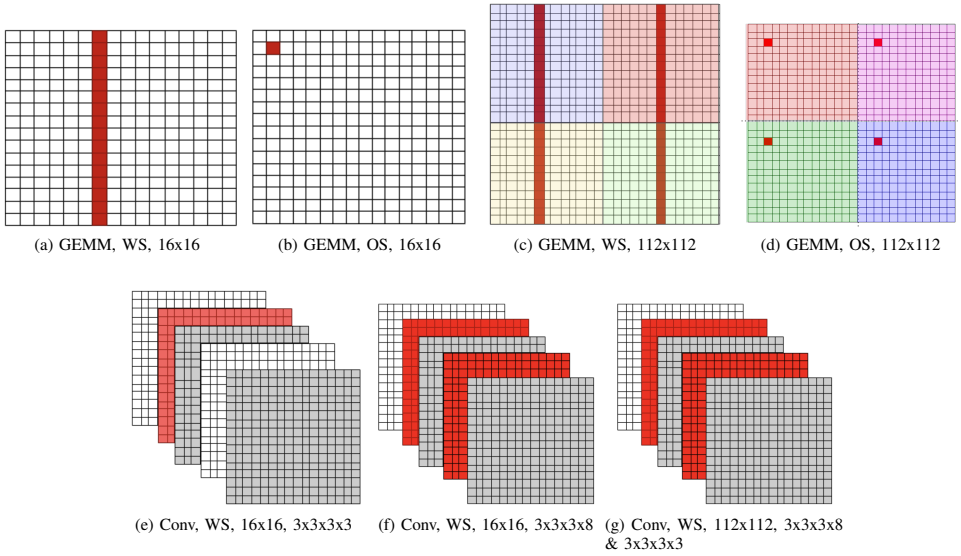
For each RQ, we ran 256 Fault Injection (FI) campaigns to exhaustively inject a stuck-at fault into every MAC unit of the 16x16 systolic array. After each FI experiment, we analyze the resulting fault pattern manually. The fault patterns are extracted by contrasting the output of the systolic array with and without FI (ground truth), keeping all other configurations the same. Additionally, for each fault pattern, we categorize it (single-row fault, single-element fault, single-column fault, etc.) based on the spatial distribution of the faults in the array.

C. Evaluation Setup

We ran RTL-level FI experiments using Amazon Web Services (AWS)'s EC2 F1 instances. F1 instances provide industrial-grade FPGAs to synthesize our Register-Transfer Level (RTL) FI framework [9]. Using FPGAs instead of simulators allowed us to perform extensive FI campaigns.

IV. Fault Injection Results and Discussion

We organize the results of the FI experiments by the RQs.



(a) GEMM, WS, 16x16 (b) GEMM, OS, 16x16 (c) GEMM, WS, 112x112 (d) GEMM, OS, 112x112

(e) Conv, WS, 16x16, 3x3x3x3 (f) Conv, WS, 16x16, 3x3x3x8 (g) Conv, WS, 112x112, 3x3x3x8 & 3x3x3x3

Fig. 3: Fault patterns corresponding to different configurations of RQ1. Figure 3a and Figure 3b corresponds to RQ1. For RQ2, contrast Figure 3a with Figure 3e, Figure 3f and contrast Figure 3c with Figure 3f and Figure 3g. Similarly, for RQ3, contrast Figure 3a with Figure 3c, Figure 3b with Figure 3d, and Figure 3e with Figure 3f and Figure 3g. For subfigures a,b,c and d, the caption is a tuple of three elements: $\langle$ Type of operation, Type of Data Flow, and Size of GEMM $\rangle$. For subfigures e,f, and g, the caption is a tuple of four elements: $\langle$ Type of operation, Type of Data Flow, Size of the input matrix, convolution kernel size ($R \times S \times C \times K$, refer Section II-B for notations) $\rangle$. Different tiles are highlighted with different colors.

A. *Effect of different parameters on Fault Patterns*

1) RQ1: Data flow mapping schemes: Figure 3b and Figure 3a show the difference between the fault patterns observed for OS and WS data flow schemes, respectively. For OS, a single fault corrupts just a single output element, while for WS, a single fault ends up corrupting an entire column in the output of the GEMM operation.

2) RQ2: Type of operation: Figure 3a and Figure 3e show the difference in the fault patterns for GEMM and convolution, respectively, with WS data mapping scheme. For GEMM, a single fault ends up corrupting the entire column of the output matrix, but for the convolution, a single fault corrupts the entire output channel. As explained in Section II-B, the convolution is implemented as a big GEMM by reshaping the input data and the convolution kernel. The resulting matrix has the dimension of $NPQ \times K$ (refer Section II-B for notations), where each output channel is mapped to each column of the systolic array. It is due to this mapping that the entire channel of the convolution is corrupted.

3) RQ3: Size of Operation: For GEMM, we considered two input sizes: 16×16 and 112×112, the first one being equal to the size of the systolic array and the latter one being larger than the systolic array. As described in Section II-C, when the input is larger than the systolic array, the operation gets broken down into small chunks (a.k.a. Tiles).

For the GEMM operation, Figure 3a and Figure 3c show the difference in fault patterns corresponding to different input sizes. Similarly, Figure 3b and Figure 3d show the difference in fault patterns, but for the OS data mapping scheme. For Figure 3c and Figure 3d, different tiles are highlighted with different colors. We observed that due to the tiling effect (when the operation size is bigger than the systolic array size), the same fault appears across multiple tiles, irrespective of the data mapping scheme. This result is intuitive since the same faulty MAC unit is used for computation across multiple tiles.

For the convolution operator, contrast Figure 3e with Figure 3f and Figure 3g. Similar to the fault patterns in the GEMM operation, the tiling effect in the convolution operation occurs when the resulting 2-D matrices - after flattening convolution into a GEMM operation, as described in Section II-C - exceeds the size of the systolic array. Due to the tiling effect, a single fault causes the corruption of multiple output channels, resulting in identical fault patterns in Figure 3f and Figure 3g.

Discussion. We found that the fault patterns vary according to the hardware and software configurations. However, all the fault patterns we found are well-defined i.e., they belong to one of the six classes: single-element corruption (e.g., Figure 3b), single-element multi-tile corruption (e.g., Figure 3d), single-column corruption (e.g., Figure 3a), single-column multi-tile corruption (e.g., Figure 3c), single-channel (e.g., Figure 3e), and multi-channel corruption (e.g., Figure 3f).

For each configuration and all of its FI experiments (one for each MAC unit), we found the same fault pattern class, regardless of the MAC unit into which we injected the fault. Moreover, the fault patterns are deterministic i.e., given the hardware configurations (size of systolic array, data mapping scheme), type of operation and its properties (like convolution and its kernel, input size), and the location of the stuck-at-fault,

we can predict the fault patterns, after taking into account the tiling effect and flattening of convolutions into GEMM.

Our findings about the well-defined fault pattern classes and their deterministic property are useful for enabling application-level fault injectors like TensorFI and LLTFI to do precise error simulations pertaining to the systolic array hardware model. RTL-level FI, although accurate, has scalability restrictions: due to limited system logic cells on current industrial-grade FPGAs, it is not feasible to synthesize and experiment with larger systolic arrays (like 128×128). Moreover, despite our use of FPGAs, each FI experiment took approximately 45 seconds (for GEMM) and 130 seconds (for Convolution), which resulted in a total of 49 hours for the FI campaigns.

Application-level fault injectors, if supplemented with a precise fault model - the fault patterns, in our case - can be used to bridge this gap and run FI campaigns even with larger systolic array sizes. Specifically to improve the precision of FI, application-level fault injectors, like LLTFI, can leverage our insights about the tiling effect and flattening of convolution operators to derive fault patterns on the fly for various systolic array sizes and data mapping schemes, as opposed to hard-coding the abstract fault pattern classes or ignoring them.

Our observation about the symmetry of fault patterns - i.e., the fault pattern class remains the same irrespective of the position of the faulty MAC unit - can also be used by application-level FIs to reduce the number of FI experiments.

V. Related Work

Characterization of hardware faults in prior work is often limited to the impact on accuracy without analyzing the fault patterns induced at the software level. To the best of our knowledge, Rech et al.'s recent work on analyzing the resilience of EdgeTPU [23] is the only one that analyzes the fault patterns induced at the software level occurring due to transient faults. We extend their work by considering other factors that might affect the manifestation of fault patterns.

Transient Faults in Systolic Array Zhang et al. [28] and Holst et al. [11] studied the effects of timing faults in systolic arrays, thus, degrading DNN's accuracy. However, their work is not directly comparable to ours due to the fundamental difference in our fault models (bitflips vs. timing errors).

Other work like that of Kundu et al. [15], Zhang et al. [29], Ren et al. [24], and Feng et al. [8] aim to understand the effects of transient faults on systolic arrays. Kundu et al. [15] analyzed the manifested faults' boundaries while leaving details of error patterns in the intermediate outputs unexplored. Ren [24] studied the geometrical pattern of faults in systolic arrays on-chip. However, it does not study the geometric pattern of manifested faults in software. Feng et al. [8] extended the characterization beyond accuracy by also considering the crash rate as a metric. However, all of these papers use software simulation of systolic arrays and do not consider fault patterns. Software-level systolic array simulations create an abstract model of the systolic array that abstracts out hardware configurations like data flow mapping schemes. In contrast, we use RTL-level FI, which captures hardware details of TPUs, thereby resulting in more accurate fault patterns.

In very recent work, Tyagi et al. [26] worked towards quantifying the accuracy of DNN accelerators under transient faults. They used RTL-level FI and proposed a new, more-accurate reliability assessment metric. However, they do not consider fault patterns emanating at the intermediate layers of the DNNs. Li et al. [17] studied the error propagation with regard to some architectural parameters of CNNs. However, in systolic arrays, the manifestation of faults varies with the hyper-parameters that were not considered in this study. Moreover, Burel et al. [4] found that output stationery (OS) dataflow is more fault-tolerant than weight stationery (similar result to ours) and thus, proposed OS-based, fault-tolerant systolic array architecture. However, Burel et al. [4] did not consider the effect of different parameters (including data mapping schemes) on the fault patterns: they only considered the final DNN accuracy degradation in the presence of faults.

Fault mitigation. Several techniques have been proposed to mitigate the effect of permanent hardware faults. Majumdar et al. [20] proposed the use of time redundancy to achieve fault resilience. Burel et al. propose an off-lining technique to detect and disable faulty MAC units [5]. We believe software-level fault characterization will enable generic software resilience solutions; instead that can be easily integrated with existing applications irrespective of the DNN accelerator being used.

To summarize, there is limited prior work on the extensive characterization of faults in systolic arrays. Among those, most of them were limited to only transient faults, that too, they either did not consider fault patterns at all, or they used an abstract software model for FI experiments, or even both. Moreover, none of the prior work that looked into fault patterns considered other essential factors like Data mapping schemes and tiling. In this work, we bridge this gap by using RTL-level FI and considering the effect of various hardware and software configurations on the resulting fault patterns.

VI. Conclusion

In this work, we studied the fault patterns in the intermediate layers of DNNs arising due to stuck-at faults within the MAC units of the systolic arrays. Moreover, we proposed an RTL-level FI framework, using which, we ran FI campaigns to understand the effect of different hardware (like data mapping schemes) and software configurations (like type and properties of tensor operations) on the observed fault patterns. We found the fault patterns to be deterministic and well-defined. We further classified these fault patterns into classes based on the spatial distribution of the faults. Our classification of fault patterns can enable application-level fault injectors (such as LLTFI) to perform more precise FI campaigns with the systolic array hardware model. This is a direction for future work.

Acknowledgments

This work was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), and a gift from Intel Research. We thank Aastha Mehta, Patrik Omland, and Michael Paulitsch for providing valuable insights.

REFERENCES

[1] System architecture of cloud tpu. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm, 2022.

[2] Udit Kumar Agarwal, Abraham Chan, and Karthik Pattabiraman. Lltfi: Framework agnostic fault injection for machine learning applications (tools and artifact track). In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 286–296. IEEE, 2022.

[3] V Agarwal and A Fung. Multiple fault testing of large circuits by single fault test sets. *IEEE Transactions on Circuits and Systems*, 28(11):1059–1069, 1981.

[4] Stéphane Burel, Adrian Evans, and Lorena Anghel. Mozart: Masking outputs with zeros for architectural robustness and testing of dnn accelerators. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–6. IEEE, 2021.

[5] Stéphane Burel, Adrian Evans, and Lorena Anghel. Mozart: Masking outputs with zeros for architectural robustness and testing of dnn accelerators. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–6, 2021.

[6] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[7] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. Serving dnns in real time at datacenter scale with project brainwave. *iEEE Micro*, 38(2):8–20, 2018.

[8] Xianglong Feng, Mengmei Ye, Ke Xia, and Sheng Wei. Runtime fault injection detection for fpga-based dnn execution using siamese path verification. volume 2021-February, 2021.

[9] Hasan Genc, Ameer Haj-Ali, Vighnesh Iyer, Alon Amid, Howard Mao, John Wright, Colin Schmidt, Jerry Zhao, Albert Ou, Max Banister, et al. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *arXiv preprint arXiv:1911.09925*, 3:25, 2019.

[10] John P Hayes. A nand model ror fault diagnosis in combinational logic networks. *IEEE Transactions on Computers*, 100(12):1496–1506, 1971.

[11] Stefan Holst, Lim Bumun, and Xiaoqing Wen. Gpu-accelerated timing simulation of systolic-array-based ai accelerators. In *2021 IEEE 30th Asian Test Symposium (ATS)*, pages 127–132, 2021.

[12] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[13] Rubens Luiz Rech Junior and Paolo Rech. Reliability of google's tensor processing units for convolutional neural networks. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pages 25–27. IEEE, 2022.

[14] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, page 615–629, New York, NY, USA, 2017. Association for Computing Machinery.

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

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[18] Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. Tensorfi: A configurable fault injector for tensorflow applications. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 313–320, 2018.

[19] Abdulrahman Mahmoud, Neeraj Aggarwal, Alex Nobbe, Jose Rodrigo Sanchez Vicarte, Sarita V Adve, Christopher W Fletcher, Iuri Frosio, and Siva Kumar Sastry Hari. Pytorchfi: A runtime perturbation tool for dnns. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 25–31. IEEE, 2020.

[20] A. Majumdar, C.S. Raghavendra, and M.A. Breuer. Fault tolerance in linear systolic arrays using time redundancy. *IEEE Transactions on Computers*, 39(2):269–276, 1990.

[21] Edward J McCluskey and Chao-Wen Tseng. Stuck-fault tests vs. actual defects. In *Proceedings International Test Conference 2000 (IEEE Cat. No. 00CH37159)*, pages 336–342. IEEE, 2000.

[22] Alessandra Nardi and Antonino Armato. Functional safety methodologies for automotive applications. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 970–975. IEEE, 2017.

[23] Rubens Luiz Rech and Paolo Rech. Reliability of google's tensor processing units for embedded applications. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 376–381. IEEE, 2022.

[24] Jiajun Ren. *Geometric characterizations of fault patterns in linear systolic arrays.* PhD thesis, Carleton University, 1994.

[25] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

[26] Abhishek Tyagi, Yiming Gan, Shaoshan Liu, Bo Yu, Paul Whatmough, and Yuhao Zhu. Thales: Formulating and estimating architectural vulnerability factors for dnn accelerators. *arXiv preprint arXiv:2212.02649*, 2022.

[27] Shuohang Wang, Yang Liu, Yichong Xu, Chenguang Zhu, and Michael Zeng. Want to reduce labeling cost? gpt-3 can help. *arXiv preprint arXiv:2108.13487*, 2021.

[28] Jeff Zhang, Zahra Ghodsi, Siddharth Garg, and Kartheek Rangineni. Enabling timing error resilience for low-power systolic-array based deep learning accelerators. *IEEE Design Test*, 37(2):93–102, 2020.

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

[30] Jeff Jun Zhang, Tianyu Gu, Kanad Basu, and Siddharth Garg. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *2018 IEEE 36th VLSI Test Symposium (VTS)*, pages 1–6. IEEE, 2018.